

1-1-2003

Determining the effectiveness of deceptive honeynets

Nirbhay Gupta
Edith Cowan University

Follow this and additional works at: <https://ro.ecu.edu.au/theses>



Part of the [Other Computer Engineering Commons](#)

Recommended Citation

Gupta, N. (2003). *Determining the effectiveness of deceptive honeynets*. <https://ro.ecu.edu.au/theses/1303>

This Thesis is posted at Research Online.
<https://ro.ecu.edu.au/theses/1303>

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Determining the Effectiveness of Deceptive Honeynets

by

Nirbhay Gupta

BSc. (Hons.)

A dissertation submitted in partial fulfilment of
the requirements for the Award of

Masters of Science (Computer Security)

At the School of Computer and Information Science
Faculty of Computing, Health and Science



**EDITH COWAN
UNIVERSITY**

PERTH WESTERN AUSTRALIA

July 2003

Abstract

Over the last few years, incidents of network based intrusions have rapidly increased, due to the increase and popularity of various attack tools easily available for download from the Internet. Due to this increase in intrusions, the concept of a network defence known as Honeypots developed. These honeypots are designed to ensnare attackers and monitor their activities. Honeypots use the principles of deception such as masking, mimicry, decoying, inventing, repackaging and dazzling to deceive attackers.

Deception exists in various forms. It is a tactic to survive and defeat the motives of attackers. Due to its presence in the nature, deception has been widely used during wars and now in Information Systems.

This thesis considers the current state of honeypot technology as well as describes the framework of how to improve the effectiveness of honeypots through the effective use of deception. In this research, a legitimate corporate deceptive network is created using Honeyd (a type of honeypot) which is attacked and improved using empirical learning approach. The data collected during the attacking exercise were analysed, using various measures, to determine the effectiveness of the deception in the honeypot network created using honeyd. The results indicate that the attackers were deceived into believing the honeynet was a real network which instead was a deceptive network.

DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

- i. incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;*
- ii. contain any material previously published or written by another person except where due reference is made in the text; or*
- iii. contain any defamatory material.*

Signature

A black rectangular box redacting the signature.

(Nirbhay Gupta)

Date

July 2003

Table of Contents

ABSTRACT	2
ACKNOWLEDGEMENTS.....	8
1. INTRODUCTION	9
2. LITERATURE REVIEW	11
2.1 DENIAL AND DECEPTION	11
2.1.1 Definitions	11
2.1.2 The Process of Deception.....	12
2.1.3 Structure of Deception	14
2.1.4 Deception in Nature	16
2.1.5 Military Deception	17
2.1.6 Computer Deception	20
2.2 OVERVIEW OF NETWORK MODELS AND PROTOCOLS.....	26
2.2.1 ISO 7 Layer Network Model.....	26
2.2.2 TCP/IP Internet Layering Model	28
2.2.3 IP: Internet Protocol	31
2.2.3.1 IP Header	31
2.2.3.2 IP Routing	33
2.2.4 TCP: Transmission Control Protocol	35
2.2.4.1 TCP Header.....	36
2.2.4.2 Connection and Termination	37
2.2.4.3 TCP State Machine	39
2.2.5 Security Issues with TCP/IP.....	41
2.2.5.1 TCP Sequence Number Prediction.....	41
2.2.5.2 ICMP attack	42
2.2.5.3 SMURF Attack	43
2.2.6 TCP/IP Fingerprinting.....	45
2.2.6.1 Classical Methods	45
2.2.6.2 TCP/IP Fingerprinting with Nmap v 3.0	46
2.3 HONEYPOTS	50
2.3.1 What are Honeypots and Honeynet?.....	50
2.3.2 Value of Honeypots	52
2.3.3 Types of Honeypots	55
2.3.3.1 BackOfficer Friendly (BOF)	55
2.3.3.2 Specter.....	55
2.3.3.3 Homemade Honeypots.....	56
2.3.3.4 Honeyd	56
2.3.3.5 Mantrap.....	57
2.3.4 Concept of Honeyd.....	59
2.3.4.1 Honeyd Architecture	60
2.3.4.2 Personalities	61
2.3.4.3 Libraries	62
2.3.5 Methods for deployment of Honeypots	62
2.3.5.1 Deception Services.....	62
2.3.5.2 Weakened Systems.....	63
2.3.5.3 Hardened Systems	64
2.3.5.4 User Mode Servers	65
2.3.6 How Honeynet is created?	66
2.3.6.1 Data Control.....	66
2.3.6.2 Data Capture	67
2.3.7 Conclusion.....	68
3. METHODOLOGY	69
3.1 INTRODUCTION	69
3.2 LITERATURE REVIEW ON RESEARCH DESIGN	69
3.3 WHEN ARE LABORATORY EXPERIMENTS USED?	72
3.4 RESEARCH PROCESS	74
3.5 ELEMENTS OF THE LABORATORY EXPERIMENTS	79

3.6	RATIONALE OF THE RESEARCH.....	80
3.7	EVALUATION OF RESEARCH METHODOLOGY.....	82
3.8	CONCLUSION	83
4.	TOOLS FOR DATA COLLECTION AND ANALYSIS	85
4.1	SNORT.....	85
4.2	MYSQL	87
4.3	WEBMIN	88
4.4	ACID (ANALYSIS CONSOLE FOR INTRUSION DATABASES).....	89
4.5	SYSLOG-NG.....	90
4.6	ETHERREAL	91
4.7	TCPDUMP	92
4.8	NMAP.....	92
4.9	GFI LANGUARD NETWORK SECURITY SCANNER.....	93
4.10	ANALYST NOTEBOOK 6	94
4.11	CONCLUSION	95
5.	OPERATING SYSTEM (OS) FINGERPRINTING	96
5.1	OVERVIEW.....	96
5.2	METHODOLOGY	96
5.3	HONEYD CONFIGURATION	98
5.4	TEST RESULTS	100
5.5	DISCUSSION	101
5.6	CONCLUSION	102
6.	THE HONEYPOT IMPLEMENTATION	103
6.1	GOALS	103
6.2	SELECTING A HONEYPOT	105
6.2.1	<i>Interaction Level</i>	<i>105</i>
6.2.2	<i>Commercial versus Homemade or freeware Solutions</i>	<i>108</i>
6.2.3	<i>Operating/System Platform</i>	<i>109</i>
6.3	DETERMINING THE NUMBER AND LOCATION OF HONEYPOTS.....	110
6.4	CONCLUSION	116
7.	FIRST TEST RESULTS ON HONEYD 0.4A.....	117
7.1	OVERVIEW.....	117
7.2	ACID ANALYSIS.....	117
7.3	ETHERREAL ANALYSIS	126
7.4	NESSUS ANALYSIS	128
7.5	FEEDBACK FROM ATTACKERS	132
7.6	IMPLICATIONS.....	133
7.7	CONCLUSION	134
8.	SECOND TEST RESULTS ON HONEYD 0.5.....	135
8.1	OVERVIEW FROM FIRST TEST RESULTS.....	135
8.2	SECOND TEST RESULTS	138
8.3	FINDINGS FROM LOG FILES.....	147
8.4	ETHERREAL ANALYSIS	148
8.5	ANALYST NOTEBOOK 6 GRAPHS	151
8.6	FEEDBACK FROM ATTACKERS	152
8.7	IMPLICATIONS.....	152
9.	THIRD TEST RESULTS ON HONEYD 0.5	154
9.1	OVERVIEW FROM SECOND TEST RESULTS.....	154
9.2	THIRD TEST RESULTS	154
9.3	FINDINGS FROM LOG FILES.....	162
9.4	ETHERREAL ANALYSIS	162
9.5	ANALYST NOTEBOOK 6 GRAPHS	166
9.6	IMPLICATIONS.....	166

10.	DISCUSSION AND CONCLUSIONS	167
11.	REFERENCES	172
APPENDIX A		181
SCREENSHOTS		181
A.1	ACID	181
A.2	Analyst Notebook 6	182
A.3	GFI LANguard	182
A.4	Ethereal	183
A.5	Webmin	183
A.6	Nmap	184
APPENDIX B		185
GNU GENERAL PUBLIC LICENSE		185
APPENDIX C		194
SCRIPTS		194
C.1	honeyd.conf (version Honeyd0.4a, used for 1 st Test)	194
C.2	honeyd.conf (version honeyd05a, used for 2 nd and 3 rd Tests)	195
C.3	web.sh	197
C.4	web98.sh	198
C.5	snort.conf	199
C.6	ftp.sh	211
C.7	create_db.sql	214
C.8	router_telnet.pl	217
APPENDIX D		219
PAPER PUBLISHED		219
APPENDIX E		220
CONTENTS OF CD		220

List of Figures

FIGURE 2.1:	THE DECEPTION PLANNING PROCESS	13
FIGURE 2.2:	STRUCTURE OF DECEPTION	14
FIGURE 2.3:	THE ISO 7-LAYER REFERENCE MODEL FOR PROTOCOL SOFTWARE	26
FIGURE 2.4:	TCP/IP PROTOCOL STACK	29
FIGURE 2.5:	IP HEADER STRUCTURE	32
FIGURE 2.6:	TCP HEADER	36
FIGURE 2.7:	TCP CONNECTION	38
FIGURE 2.8:	TCP CONNECTION TERMINATION PROCESS	39
FIGURE 2.9:	TCP STATE MACHINE	40
FIGURE 2.10:	SMURF ATTACK	43
FIGURE 2.11:	NETWORK DIAGRAM OF A HONEYPOT DEPLOYED ON A DMZ TO DETECT ATTACKS	51
FIGURE 2.12:	THE HONEYNET	52
FIGURE 2.13:	COMPLEXITY VS. PERFORMANCE GRAPH OF VARIOUS HONEYPOTS	58
FIGURE 2.14:	TRAFFIC TO HONEYD AND ITS VIRTUAL HONEYPOTS	59
FIGURE 2.15:	HONEYD ARCHITECTURE	61
FIGURE 2.16:	USER MODE SERVERS	65
FIGURE 3.1:	THEORETICAL FRAMEWORK: STAGES OF RESEARCH PROCESS	74
FIGURE 4.1:	TOOLS FOR DATA COLLECTION AND ANALYSIS	85
FIGURE 4.2:	SNORT OVERVIEW	86
FIGURE 5.1:	TCP/IP FINGERPRINTING RESULTS USING NMAP	101
FIGURE 6.1:	DEPLOYMENT OF HONEYPOT IN A LABORATORY NETWORK STRUCTURE	112
FIGURE 6.2:	THE VIRTUAL HONEYNET	113
FIGURE 6.3:	ARCHITECTURE OF DATA COLLECTION MACHINE	116
FIGURE 7.1:	ACID ANALYSIS	117

FIGURE 7.2: TRAFFIC PROFILE BY PROTOCOL USING ACID	125
FIGURE 7.3: PROTOCOL HIERARCHY STATISTIC OF TCPDUMP.LOG.1045547976	126
FIGURE 7.4: MOST DANGEROUS SERVICES ON THE NETWORK	129
FIGURE 7.5: SERVICES THAT ARE THE MOST PRESENT ON THE NETWORK	129
FIGURE 7.6: MOST DANGEROUS HOST ON TO THE NETWORK	131
FIGURE 8.1: IMPROVED HONEYNET ARCHITECTURE	137
FIGURE 8.2: ACID ANALYSIS	138
FIGURE 8.3: TRAFFIC PROFILE BY PROTOCOL USING ACID	146
FIGURE 8.4: PROTOCOL HIERARCHY STATISTICS OF TCPDUMP.LOG.1048208266	148
FIGURE 8.5: PROTOCOL HIERARCHY STATISTICS OF TCPDUMP.LOG.1048298249	150
FIGURE 9.1: ACID ANALYSIS	154
FIGURE 9.2: TRAFFIC PROFILE BY PROTOCOL USING ACID	160
FIGURE 9.3: PROTOCOL HIERARCHY STATISTICS OF TCPDUMP.LOG.1053944062	163
FIGURE 9.4: UDP PACKET	164
FIGURE 9.5: PACKET CAPTURED SHOWING SCRIPT COMMAND USED TO RETRIEVE INFORMATION	164
FIGURE 9.6: PROTOCOL HIERARCHY STATISTICS OF TCPDUMP.LOG. 1054023963	165

List of Tables

TABLE 2.1: SUMMARY OF DIFFERENT TYPES OF HONEYPOTS	58
TABLE 3.1: RESEARCH METHODOLOGIES	71
TABLE 3.2: RESEARCH METHODS	71
TABLE 5.1: NMAP SCANNING RESULTS	100
TABLE 6.1: HONEYPOT LEVEL OF INTERACTION	108
TABLE 6.2: THE HONEYPOT SYSTEM CONFIGURATION	112
TABLE 6.3: SUMMARY OF IP ADDRESSES ALLOCATED ON CORPORATE NETWORK	114
TABLE 6.4: DATA COLLECTION MACHINE SPECIFICATION	115
TABLE 7.1: 11 ALERT CATEGORIES	118
TABLE 7.2: MOST 5 FREQUENT ALERTS	124
TABLE 7.3: LIST OF PORT NUMBERS WITH OCCURRENCE OF ALERTS	125
TABLE 7.4: LIST OF HOSTS WITH COMMON SECURITY HOLE ON PORT 80	131
TABLE 8.1: SUMMARY OF IP ADDRESSES ALLOCATED ON CORPORATE NETWORK	137
TABLE 8.2: 14 ALERT CATEGORIES	139
TABLE 8.3: MOST 5 FREQUENT ALERTS	146
TABLE 8.4: LIST OF POPULAR DESTINATION PORT NUMBERS WITH OCCURRENCE OF ALERTS	147
TABLE 9.1: 12 ALERT CATEGORIES	155
TABLE 9.2: MOST 5 FREQUENT ALERTS	159
TABLE 9.3: LIST OF POPULAR DESTINATION PORT NUMBERS WITH OCCURRENCE OF ALERTS	161

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor Mr. Craig Valli for his continuous guidance and support during the research and preparation of this dissertation. His inspiring and encouraging supervision has guided me to a deeper understanding of research work. Also, his valuable comments during the entire work have made this dissertation possible.

I am grateful to Assoc. Prof. William Hutchinson for his expert comments on methodological and philosophical aspects of this dissertation. I would also like to thank Lecturer Justin Brown for reviewing my thesis before the submission. I also want to thank rest of the Computer Security Research Group members who time to time provided me with their valuable feedback and suggestions during the weekly meetings.

Special thanks to all the participants who participated in this research. Their support and cooperation made this research a success.

Finally, I would like to thank my parents and friends for their continuous support and encouragement during the entire course of my study in Australia.

1. Introduction

In the last few years, the networking revolution has finally come of age. The Internet is purported to provide limitless possibilities and opportunities but on the other side also increases the risks of malicious intrusions by hackers or malicious attackers community (Sundaram, 2001). In the rest of the thesis, the term 'hackers' will be used to refer to malicious attackers.

It is very important to understand, design and implement some form of security mechanism to prevent or detect unauthorised users. These intrusions can be detected. Due to the increased attempts of intrusion into systems, the concept of 'honeypot' systems was developed (Spitzner, 2002). Honeypots are used to trap and decode attack methods used by the hackers (Brenton, n.d; Klug, 2000; Spitzner, 2002). Honeypots can provide a deceptive defence mechanism in which the attackers are deceived into believing they are intruding into a real production system. The correct deployment, monitoring and analysis of honeypots help in increasing our understanding of attackers' modes of operations and tools in details.

Deception can be referred as the state of being deceived or misled (Webster's Revised Unabridged Dictionary, 1998). It can be considered as creation of false environment to fool or deceive people. Honeypot systems are meant to create a false computing environment in order to keep away the attackers from the real network environment and entrap them in a false system. Honeypots are designed to audit the activity of intruder, save log files and record events like processes started, changes or deletion of files and key strokes.

The significance of this research is to improve the defensive capability of Honeynet networks. This research is significant to universities, government organisations, educational institutions who will be able to use the data generated from this research as a platform to continue with further research into computer and network security. Private organisations or individuals can use this research for deploying this or similar network architecture into their networks with the aim of making it more secure. The outcomes of this research will allow the organisations and individuals to understand

and learn about various, unfamiliar attacks which may be captured on the implemented honeynet.

The purpose of this research is to improve the level of deception presented to attackers in a honeypot design. To achieve a higher level of deception the designed deception should be regularly improved or updated with new deceptive services. This can be achieved by hardening the deceptive honeypots and testing the effectiveness of the deception using empirical learning approach. The empirical learning approach means by testing the systems in a cyclic manner where results of each tests depends on the other. This will help us in improving the ability of the deceptive honeypots to gather attack intelligence while the network is being probed or attacked.

Following are the two main research questions outlined to achieve the desired purpose of the research:

1. What factors affect the effectiveness of deception in network environments when using honeypot systems?
2. How can these factors be enhanced to perform a more effective level of deception?

This thesis starts with a detailed literature review in chapter 2. This literature review includes the concepts of deception in everyday life to cyberspace, detailed description about TCP/IP and its security issues and the concept of honeypots. Chapter 3 presents the detailed methodology adopted by the researcher. Chapter 4 discusses various tools used for data collection and analysis purpose. Operating system fingerprinting results are discussed in chapter 5.

The honeypot implementation is explained in chapter 6. Chapter 7-9 discusses the analysis of results obtained in first, second and third testing respectively. Chapter 10 presents the final discussion and conclusions. The included CD with the thesis contains the charts generated by Analyst Notebook 6 from using the data from honeyd log files.

2. Literature Review

2.1 Denial and Deception

This section of the chapter discusses the role of denial and deception in various forms. Section 2.1.1 defines the terms “denial” and “deception”. Section 2.1.2 describes the process of deception used for implementing a successful deception. Section 2.1.3 discusses the structure of deception. Section 2.1.4 describes how deception exists in various forms in nature. Section 2.1.5 describes the role of deception in military. Finally, section 2.1.6 describes the role of deception in information systems or computers.

2.1.1 Definitions

There are many different ways by which “denial” can be defined. According to Shulsky (Godson & Wirtz, 2002, p15), “denial” refers to the attempt to block all information channels by which an enemy could learn some truth, thus preventing him or her from reacting in a timely manner. For our purposes, it can be referred as safeguarding the information (digital information) from an adversary.

“Deception” by contrast is defined as the deliberate alteration of data or a situation's context to promote a desired outcome (Hutchinson & Warren, 2002). It is an attempt to cause the person to believe something which is not true.

As reported by Gerwehr and Glenn (2000), Whaley (1969) has defined deception as “information designed to manipulate the behaviour of others by inducing them to accept a false or distorted presentation of their environment – physical, social or political.” Deception exists not only among humans but also in animals and plants. In the animal and plant kingdoms, deception is among the most effective and widespread tools for survival. For example, when confronted by a predator, the *Sepioida* squid inserts a cloud of ink, which is colored and shaped just like the squid, between itself and the predator. The squid then changes color and darts away, leaving a confused predator in its wake (Gerwehr & Glenn, 2000).

Deception is a tactic to survive and to defeat the motives of attackers. Cohen, et al. (2001) quotes from Sun Tzu that “*“All warfare is based on deception”*”, which means that all warfare *should* be based on deception. Deception is not new to warfare. Even the great epics describe deception having been used in wars. Mythology talks about it every time. From the death of Hercules to the defeat of the Spanish Armada to World War II to Operation Desert Storm, deception has been used widely.

2.1.2 The Process of Deception

The process of deception is in many ways, like a chess game. If the person doing the deceiving is more skilled (or simply does a better job) than the person being deceived, then the deception will be successful. In deception, (just as in chess), some encounters will end in “stalemate”.....in that the person being deceived did not get enough information to really determine that a deception had occurred.

The actual deception process requires 3 main elements as defined by Gerwehr & Glenn (2000):

- Objective
- Target
- Story

Initially to have a successful deception, the *objective* to create that deception should be very clear in mind. This could range from simple survival to gaining strategic surprise. The next step should be to have an idea of what the deceiver wants the enemy to do in order to achieve that objective. This could be as simple as getting the enemy to focus on location A instead of location B. For instance, in a Honeypot (Spitzner, 2003) environment the objective of the deceiver is to create a deceptive network so that the malicious attacker is kept away from the main production network.

After identifying the objective for creating the deception, the next step is to identify the *target* of the deception. After the target is identified, the deceiver parlays intelligence about the target into a profile of that person’s beliefs, intentions and

capabilities. A well-constructed deception is built around that intelligence and exploits it. In a Honeypot environment, the network should resemble a normal corporate network with some common vulnerabilities in order to attract the malicious attackers. The objective of implementing a honeypot is to identify the intentions of the malicious attackers for example, what they intend to do once having an access to the network. Also it helps in understanding what their capabilities are, in terms of accessing and attacking the networks or hosts on the network, what type of attacks they are capable of performing such as, denial of service attacks or password cracking etc (Spitzner, 2003).

Knowing the objective and the target of deception, the deceiver should formulate a *story* that must be told to the target to produce some misperceptions. The story is told through means of deception, such as camouflage or disinformation. In a Honeypot, deceptive services and operating systems are implemented to create the misperception in the mind of the attacker and keep him or her busy in the deceptive network to safeguard the main production network (Spitzner, 2003). This network may be equipped with various hosts running multiple operating systems with some open vulnerabilities and services to attract attackers. Such deceptions need to be carefully designed so that the total deception created to attract attackers does not look suspicious. If the attackers become suspicious then they may leave the honeypot immediately and this may not resolve the purpose of the research. The complete Deception planning process is illustrated in Figure 2.1 below:

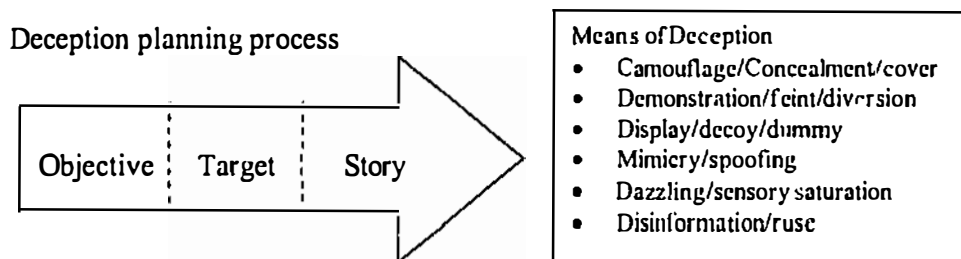


Figure 2.1: The Deception Planning Process

(Adopted from Carwehr & Glenn, 2000, p. 28)

The actual execution of the deception planning process moves in the reverse direction: informational elements being manipulated are transmitted, creating the story, in the mind of target, to achieve the objective.

2.1.3 Structure of Deception

Deception can be categorised in two types: *hiding the real* and *showing the false* (Bowyer, 1982). Hiding can be further divided into masking, repackaging and dazzling. Showing can be further divided into mimicking, inventing and decoying. The figure 2.2 below illustrates the structure of deception:

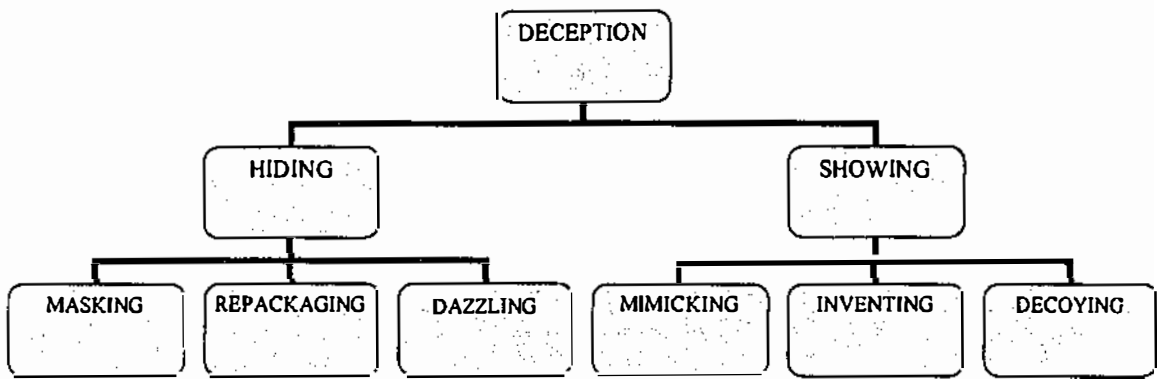


Figure 2.2: Structure of Deception

(Adopted from Bowyer, 1982)

Masking occurs when the real is hidden by blending with a background, integrating itself with surroundings. It is a type of camouflage: battleships painted grey to blend with the ocean. Steganography is a good example of masking in relation to computer and network security. Messages are embedded and hidden inside an image and it is very hard to identify if the image is encrypted with some hidden message. Repackaging occurs when the real is hidden by a new wrapping. The concept of repackaging does exist over the web (Hutchinson and Warren, 2002). Users are misled to believe which is not true. For example, most of the computer viruses are packaged into emails which appears to be genuine or with desirable contents. Like I

LOVE YOU virus which spread using the address book of the affected computer with a false love letter along with it. Such disinformation can be very destructive as Internet has become the important part of the corporate networks. Therefore it is very important to be aware and be prepared to avoid such deceptions in a network environment.

Dazzling occurs when an object can neither blend in the background by some form of masking nor be repackaged effectively then the qualities of the object may be changed in such a way as to mystify. All military and diplomatic codes and ciphers are a form of dazzling (Bowyer, 1982). Dazzling over the internet can be co-related with causing denial of service attack over a network. The victims' network resources are mainly used in coping with the attack rather than its normal operation. Dazzling can also be used for sending Trojans while causing a dazzling DoS attack.

Mimicking involves creation of a replica of reality with one or more characteristics of the real. It is similar to masking but nothing is hidden, just to hide the reality of what is there. The most famous example of mimicking occurred in Canada when an illiterate pensioner died in Toronto several years ago. His family cut off his thumb and preserved it in formaldehyde, and settled down to some years of monthly checks endorsed as usual with his thumb print. In **inventing**, false is displayed by creating an alternative reality. A false document appears real but it is not. **Decoying** openly shows something, which is not true. For example, dummy tanks used successfully by the Serbs in the recent Balkans' conflict to confuse the attack aircraft. (Bowyer, 1982; Hutchinson and Warren, 2002). All the above three, mimicking, inventing and decoying, can be extensively used in honeypots/Honeynets. As in Honeynets, a replica of real network is created and a false network environment is created which looks very real to the intruder. This network emulates various services and operating systems which resembles to the real services and operating systems.

Deception has been in existence not only among humans but also in plants and animals. In order to understand the phenomenon of deception among the Information Systems or computers, it is important to understand the origin of deception from nature which is adopted into the military and computer systems deception.

2.1.4 Deception in Nature

Every species' in the environment has its predators and its prey. They all use deception in one form or another, either to protect themselves from the predator or in search of their prey for their living. According to Bowyer (1982), deceptive behaviour and physical distortion in various species are determined by physical structure and genetic makeup that has emerged from a process of selection over the course of evolution or possibly a single moment of mutation.

Many types of deception are employed in nature such as masking, repackaging, dazzling, mimicking, inventing and decoying as illustrated in Figure 2.2.

Biological Examples of deception in nature

Masking

Certain animals have characteristic to blend in the surrounding to 'hide' them. As in case of polar bear, whose fur creates an illusion to appear white, due to the nature of reflection, to masks its presence by blending with its surrounding environment, snow (Polar Bear International, 2002)

Repackaging

Hiding from real by repackaging can be perceived in various ways and can be either dangerous or harmless. Chameleons have characteristic of changing colours of their skin surface according to the surrounding background (Carthy, 1972).

Dazzling

There are situations when masking and repackaging does not help the victims to hide themselves from the predator. In such situations, dazzling is a mode of deception which protects the victim from the attacker or predator. A good example of dazzling could be when a predator attacks an octopus. Under attack the octopus shoots out ink that dazzles the attacker and the octopus's withdraws to safety. Since the blue ink is not venomous, it temporarily impairs the smell senses of the attacker while the octopus flees to safety (Haveeru Daily, 2003).

Mimicking

Mimicking is used by animals not only to hide them but also for their own advantages as well. For example, an anglerfish looks like a rock except for a long filament waving before its mouth. The prey may lure close to the fish to investigate about the filament and assume that there is nothing but a piece of rock (Bowyer, 1982, p51). The cuckoo bird abandons its eggs in the nests of other birds. These eggs are small in size and mimic the colour of the nested bird. Therefore, the cuckoo bird's eggs are fledged by the nesting bird rather than the real biological parent and the host is deceived.

Inventing

Inventing is used by animals when mimicking is not sufficient to protect them or hide them. Further from the previous example, anglerfish also invents a new reality. Other than having a body like rock and a long filament, anglerfish simulates a small fish from a long appendage on the forehead of the female anglerfish acting as bait, motioning back and forth, and glowing from the reflection of millions of light producing bacteria (Bowyer, 1982; Doran, 2000; Monterey Bay Aquarium, 2003).

Decoying

A decoyed deception is used to distract predators from the discovered real. For example, when a bird fears for her brood, she confuses by fluttering away or decoying by looking left and running towards right, misdirecting the predator away from the nest. Once the decoy is successful in distracting the predator away from the siblings, the mother bird takes flight leaving predator away from the siblings.

2.1.5 Military Deception

Like animals, humans do need to make decisions to survive. In the realm of conflict and war, deception is both applicable and valuable. It supports in developing decoys to deflect the enemy attention away from major attacks. Military deception aims to deliberately induce misperception in another for tactical, operational, or strategic

advantage (Gerwehr & Glen, 2000). According to Joint Chiefs of Staff (JCS) Memorandum of Policy (MOP) 1 16 mentioned in Cohen (2001):

“Historically, military deception has proven to be of considerable value in the attainment of national security objectives, and a fundamental consideration in the development and implementation of military strategy and tactics. Deception has been used to enhance, exaggerate, minimize, or distort capabilities and intentions; to mask deficiencies; and to otherwise cause desired appreciations where conventional military activities and security measures were unable to achieve the desired result. The development of a deception organization and the exploitation of deception opportunities are considered to be vital to national security. To develop deception capabilities, including procedures and techniques for deception staff components, it is essential that deception receives continuous command emphasis in military exercises, command post exercises, and in training operations.”

Deception is used to adversely affect an opponent’s decision-making process. It can be employed against an entire enemy army or alternatively could be applied against anyone at all unfriendly to the deceiver.

There are many collections of information on deception in war. As mentioned by Cohen (2001) the work of Whaley (1969), which includes details of 67 military deception operations between 1914 and 1968. Dunnigan and Nofi (1995) also reviewed the history of deception in warfare and categorised them as concealment, camouflage, false and planted information, ruses, displays, demonstrations, feints, lies, and insight.

According to Fowler and Nesbitt (1995), there are six general principles for effective tactical deception in warfare (Rowe & Rothstein, 2003):

1. Deception should reinforce enemy expectations.
2. Deception should have realistic timing and duration.
3. Deception should be integrated with operations
4. Deception should be coordinated with concealment of true intentions
5. Deception realism should be tailored to needs of the settings.

6. Deception should be imaginative and creative.

Applying these principles to information systems:

Principle 1: In information systems, systems should be designed in a controlled manner with collecting intelligence about information resources and propagating the attacks to neighbouring systems. The deception should be designed in order to pretend to aid the attackers.

In information systems, honeypots are deployed to deceive the attackers and keep them busy in honeypot (Spitzner, 2003). Therefore, honeypots should be able to provide the attacker with the necessary services and features which could reflect to them as a legitimate network and thus they are able to attack this network successfully without getting suspicious.

Principle 2: Deceptions should not be too slow or too fast in compare to the activities they are suppose to stimulate. For example, in honeypots there could be some delay in data transfer from one system to another in a network environment but it should not take too much time too so that an attacker becomes suspicious or gets discouraged in accessing the system.

Principle 3: This principle states that deception should be integrated with operations. When deploying honeypots, the main operation in this context is to gather information regarding the hackers (*ibid*, 2003). Therefore, honeypots should not serve to normal genuine users but malicious attackers by encouraging them to log in and divert their resources. Recording such activities could provide intelligence about attack methods.

Principle 4: Deception should be coordinated in a careful manner. It should not give the attacker any suspicion that he or she has been deceived. For example, if from the honeypot the attacker downloads any file then all the properties of file download should be maintained in the complete file system such as directory-listing utility, file editors, file backup routines, web browser and execution monitor. This will reduce the chances of attacker getting suspicious and thus honeypot will be able to serve its purpose.

Principle 5: It is always not necessary to create or develop a complete detailed deception. For example, most of the attackers may just be interested in downloading their programs or scripts used for attacking systems and installing them, therefore it will be important to have a deceptive file-download utility such as FTP. But it may be unlikely for an attacker to archive his or her files so the archive utility need not be deceptive in nature. This may provide the simplicity to the honeypots without crowding them with unnecessary services and features. Such simplistic honeypots are also easy to maintain.

Principle 6: This principle is not very easy to implement in information systems as responses to most of the techniques used in information systems are predictable. Methods from artificial intelligence may suggest few ways to produce convincing simulated activity in creative ways.

The layers of deception are implemented with careful consideration of the adversary. Different adversaries penetrate and fall prey to deceptions in numerous ways depending upon their knowledge, experience, capabilities, determination, and resources (Gerwehr & Anderson, 2000).

2.1.6 Computer Deception

Since early 90's the use of deception in information systems have come to the main stream. Following are the few studies conducted during that period are (Cohen, 2001):

- In 1991, paper published by researchers of AT&T about creation of 'Jail' to track an attacker to observe their actions (Cheswick, n.d).
- An approach to using deceptions for defence by customizing every system to defeat automated attacks was published in 1992 (Cohen, 1992);
- Descriptions of Internet Lightning Rods in 1996. (Cohen, 1996)

Since then deception has increasingly been explored as a key technology area for innovation in information protection. There has been some follow-on-studies,

technologies and increasing adoption of technical deceptions for defence of information systems. This includes:

- **Commercial and Non-commercial Deception Products:** Development of various deception products like Deception Tool Kit (DTK), Honeyd, Mantrap and Spector.
- **The HoneyNet Project:** The Honeynet Project (Anonymous, 2002b, Spitzner, 2002) is a non-profit research organization of security professionals dedicated to information security. Its goal is to learn the tools, tactics, and motives of the blackhat community and share these lessons learned.
- **The RIDLR:** The Re-configurable Intrusion Detection Laboratory Research (RIDLR) is a project launched from Naval Post Graduate School designed to test out the value of deception for detecting and defending against attacks on military information systems
- **RAND studies:** In 1999, RAND completed an initial survey of deceptions in an attempt to understand the issues underlying deceptions for information protection.

Computers are automated machines created by humans and cannot really be called 'aware' in the sense of people. Therefore, when deception is used against computers, it is actually used against the skills of human(s) that design, program and use the computer. Computers are better in detecting the deception than people because they possess extraordinary analysis capability and the logical processes used by computers are normally quite different than the processes used by people. This provides some level of redundancy into the system.

Cohen (2001) model of computer deception is based on Cohen's "Structure of Intrusion and Intrusion Detection" (Cohen, 2000b). In this model, a computer system and its vulnerabilities are described in terms of intrusions at the hardware, device driver, protocol, operating system, library and support function, application, recursive language and meaning vs. content levels. The levels are all able to interact, but they usually interact hierarchically with each level interacting with the ones just above and below it.

This model is based on the notion that at every level of the computer's cognitive hierarchy signals can either be induced or inhibited. Deception detection and response capabilities are key issues in the ability to defend against deceptions.

Hardware Level Deceptions

The hardware of the system can be easily altered which may result in arbitrarily different behaviour than expected. Intrusion detection systems can be used to determine the improper modifications to hardware which are primarily based on built-in self-test mechanisms such as the power on self test (POST) routine in a typical personal computer. These mechanisms are designed to detect fault types in hardware not the malicious alterations. Thus deception of these mechanisms is easy to do without altering their value in detection fault type.

Intrusions can also be the result of the interaction of hardware of different sorts rather than the specific use of a particular type of hardware. Hardware-level deceptions designed to induce desired observables are relatively easy to create and hard to detect. The problem with using hardware level deception for defence against serious threat types is that it requires physical access to the target system or logical access with capabilities to alter hardware level functions.

Driver Level Deceptions

Drivers who usually have unlimited hardware access and can be installed from or by applications are nonnally ignored by intrusion detection and other security systems. A driver level of deception is capable of causing the driver to process items of interest without passing any information to other parts of the operating environment (Cohen, 2001).

From a defensive point of view, drivers are good targets. For example, a driver may be required to be installed to gain access to defended sites. Applications like RealAudio, QuickTime, HotBar require their individual drivers to be installed before anyone can access these programs. Once the target loads the required driver, hardware access is gained and exploits can be launched. This technique is offensive in nature.

Protocol Level Deceptions

Various flaws in the IP protocol and in cryptographic protocols can be exploited for intrusion purposes. For example, TCP/IP protocol is widely used for data communication therefore is most prone for attack. It carries data from one host to another so an intruder may try to exploit the vulnerabilities associated with this protocol which may give access to the victims machine or alter the data. Other than few known flaws that are part of active exploitations, most current intrusion detection systems do not detect such vulnerabilities. Therefore, the more feasible approach is to differentiate between protocols that are allowed and those that are not. This can be further improved by differentiating based on location, time, protocol type, packet size and makeup and other protocol level information. There are also interactions between hardware and protocols, therefore an exploit in hardware device may result in arbitrary behaviour of a particular protocol.

Defensive protocol level deceptions are easy to develop and hard to defeat. Deception Tool Kit (DTK) (<http://www.all.net/dtk>) and D-WALL both use protocol level of deceptions to great effect and are simple to implement. Most intelligence gathering starts at protocol level.

Operating System Level Deceptions

Since operating systems protection can be circumvented in a number of ways, therefore there are various intrusions possible at operating system (OS) level. Operating systems can have complex interactions with other operating systems in the network or environment as well as with different programs operating within OS. It may also have some complex interactions with protocols with hardware conditions, and these interactions are extremely complex to analyse. Thus deceptions based on mixed levels including the OS are likely to be undetected as deceptions (*ibid*, 2001).

Operating systems are the most common point of attack against systems today as they possess tremendous amount of cover and capability. They have unlimited access within the system and the ability to control the hardware so as to yield arbitrary external effects and observables. For example, with the knowledge of the operating system type and version, it becomes an easy target for the attacker to attack the machines using the exploits associated with that type of operating system. If an

attacker knows that the victim machine is running on Windows operating system, he or she can narrow down their attacks which are only associated with windows based machines. To have a defensive deception at the target's operating system level requires offensive action on the part of the deceiver. Thus other level of deceptions needs to be exploited to have a defensive deception on OS.

Library and Support Function Level Deception

Libraries and support functions are often embedded within a system and are largely hidden from the programmer so that their role is not as apparent as either operating system calls or application level programs. For example, programs written in C language has embedded sets of functions that automate various other functions. The high level of interaction of libraries is a symptom of the general intrusion detection problem. Many current operating systems have the ability of loading libraries as device drivers which may provide hardware controls.

Libraries functions as defensive deceptions can be of big help but due to the limitation of libraries in terms of their effectiveness they may not be able to provide a secure level of defensive deception.

Application Level Deceptions

Applications provide many new opportunities for deceptions. They always trust the data they receive so false content can be easily generated. Known attack detection tools and anomaly detection have been applied at the application level with limited success (*ibid*, 2001).

The interaction of an application level with any library level may allow remote user to cause any unexpected malicious behaviour within the system. It is common for many programmers to miss the error detection and implementations at system and library call level. This may result in unexpected halt of the application or may pass wrong values to another application. Therefore, application level defensive deceptions are major area of interest because applications tend to directly influence the decision processes made by attackers. An Application level deception might be used to cause a system that is overrun to act on the wrong data.

While “denial” and “deception” are separate terms that can be distinguished conceptually they are however closely intertwined in practice. For a successful deception to occur, it must include denial. To make an enemy believe in the cover story placed in front of him or her, information that would reveal the truth must be denied to him or her. It is may be impossible to imagine a deception effort that does not involve denial (Godson & Wirtz, 2002).

2.2 Overview of Network Models and Protocols

This section of the chapter discusses the structure of TCP/IP protocol and how the TCP/IP connection and termination is implemented. Also it discusses various vulnerabilities associated with the TCP/IP protocol.

2.2.1 ISO 7 Layer Network Model

The standard model for networking protocols and distributed applications is the International Standard Organization's Open System Interconnect (ISO/OSI) model. The model is generic and applies to all network types, not just TCP/IP, and all media types, not just Ethernet. It defines seven network layers as shown below.

7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data Link
1	Physical

Figure 2.3: The ISO 7-Layer Reference Model for Protocol Software

(Parker, 2000; Comer, 1995)

Layer 1 - Physical

Physical layer defines the cable or physical medium itself, e.g., Thinnet, Thicknet, unshielded twisted pairs (UTP). All media are functionally equivalent. The main difference is in convenience and cost of installation and maintenance. Converters from one media to another operate at this level.

Layer 2 - Data Link

Data Link layer defines the format of data on the network. A network data frame, aka packet, includes checksum, source and destination address, and data. The largest packet that can be sent through a data link layer defines the Maximum Transmission Unit (MTU). The data link layer handles the physical and logical connections to the packet's destination, using a network interface. A host connected to an Ethernet would have an Ethernet interface to handle connections to the outside world, and a loopback interface to send packets to itself

Layer 3 - Network

It provides a means for communicating open systems to establish, maintain and terminate network connections. The IP protocol lives at this layer, and so do some routing protocols such as Address Resolution Protocol (ARP). ARP is used to map the IP address to its hardware address. All the routers in a network are operating at this layer. IP is responsible for routing packets from one network to another. Network layer may have to break large packets, which are larger than their MTU, into smaller packets and the host receiving the packets may have to reassemble the fragmented packet (Comer, 1995; Parker, 2000).

Layer 4 - Transport

The transport layer is responsible for the end-to-end integrity of transmissions. Unlike the Data Link Layer, the transport layer is capable of providing this function beyond the local LAN segment. It can detect packets that are discarded by routers and automatically generate a retransmit request (Comer, 1995; Rodriguez, 2001).

Transport layer is also capable of re-sequencing of packets that may have arrived out of order. This can happen for a variety of reasons. For examples, the packets may have taken different paths through the network, or some may

have been damaged in transit. In any case, the transport layer is capable of identifying the original sequence of packets, and must put them back into that sequence before passing their contents up to the Session Layer (Comer, 1995).

Layer 5 - Session

It provides for two communicating presentation entities to exchange data with each other. Another service that is offered as a part of the Session Layer might include data synchronization. Checksums may also be included at the Session Layer as a part of data synchronization

Layer 6 – Presentation

This is where application data is packed or unpacked, ready for use by the running application. Protocol conversions, encryption/ decryption and graphics expansion all takes place here.

Layer 7 – Application

Application Layer provides the interface between those applications and the network's services. This layer can be thought of as the reason for initiating the communications session. For example, an email client might generate a request to retrieve new messages from the email server. This client application automatically generates a request to the appropriate Layer 7 protocol(s) and launches a communications session to get the needed files (Comer, 1995, Parker, 2000).

2.2.2 TCP/IP Internet Layering Model

Like most networking software, TCP/IP is modelled in layers. This layered representation leads to the term *protocol stack*, which refers to the stack of layers in the protocol suite. (Rodriguez, 2001)

By dividing the communication software into layers, the protocol stack allows for division of labour, ease of implementation and code testing, and the ability to develop alternative layer implementations. Layers communicate with those above and below via concise interfaces. In this regard, a layer provides a service for the layer directly above it and makes use of services provided by the layer directly below it. For example, the IP layer provides the ability to transfer data from one host to another without any guarantee to reliable delivery or duplicate suppression. Transport protocols such as TCP make use of this service to provide applications with reliable, in-order, data stream delivery. The below figure 2.4 shows how TCP/IP protocols are modelled in four layers (Rodriguez, 2001):

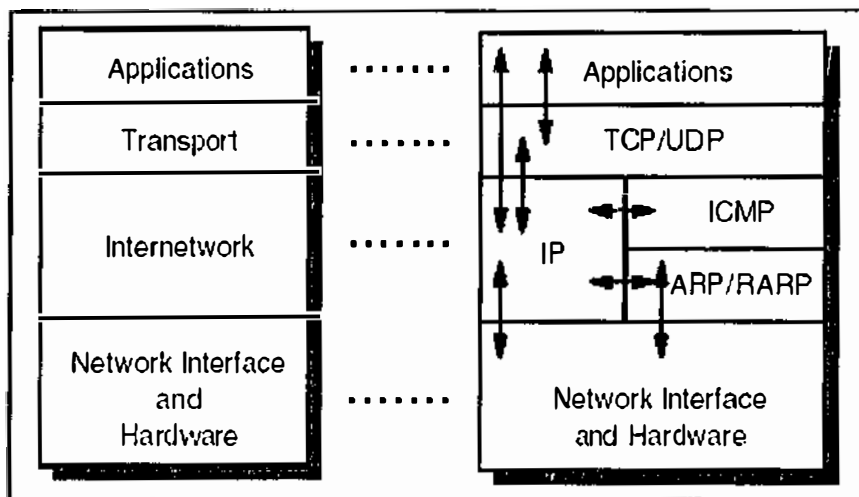


Figure 2.4: TCP/IP Protocol Stack

(Rodriguez, 2001)

Application Layer

The application layer is provided by the program that uses TCP/IP for communication. An application interacts with one of the transport level protocols to send or receive data. Each application program chooses a style of transport needed, which can be either a sequence of individual messages or a continuous stream of bytes. The application program passes data in the required form to the transport level for delivery (*ibid*, 2001).

Transport Layer

The primary duty of transport layer is to provide communication from one application program to another. Such communication is often called end-to-end. The transport layer may regulate flow of information. It may also provide reliable transport, ensuring that data arrives without error and in sequence. Multiple applications can be supported simultaneously. The transport layer must accept data from several user programs and send it to the next lower layer. Therefore, it adds additional information to each packet, including codes that identify which application program sent it and which application program should receive it, as well as a checksum (*ibid, 2001*).

Internetwork Layer

The internet layer handles communication from one machine to another. It accepts a request to send a packet from the transport layer along with an identification of the machine to which the packet should be sent. It also handles incoming datagrams, checking their validity, and uses the routing algorithm to decide whether the datagram should be processed locally or forwarded. Finally the internet layer sends ICMP error and control messages as needed and handles all incoming ICMP messages (*ibid, 2001*).

Network Interface Layer

The network interface layer, also called the link layer or the data-link layer, is the interface to the actual network hardware. This interface may or may not provide reliable delivery, and may be packet or stream oriented. In fact, TCP/IP does not specify any protocol here, but can use almost any network interface available, which illustrates the flexibility of the IP layer (*ibid, 2001*).

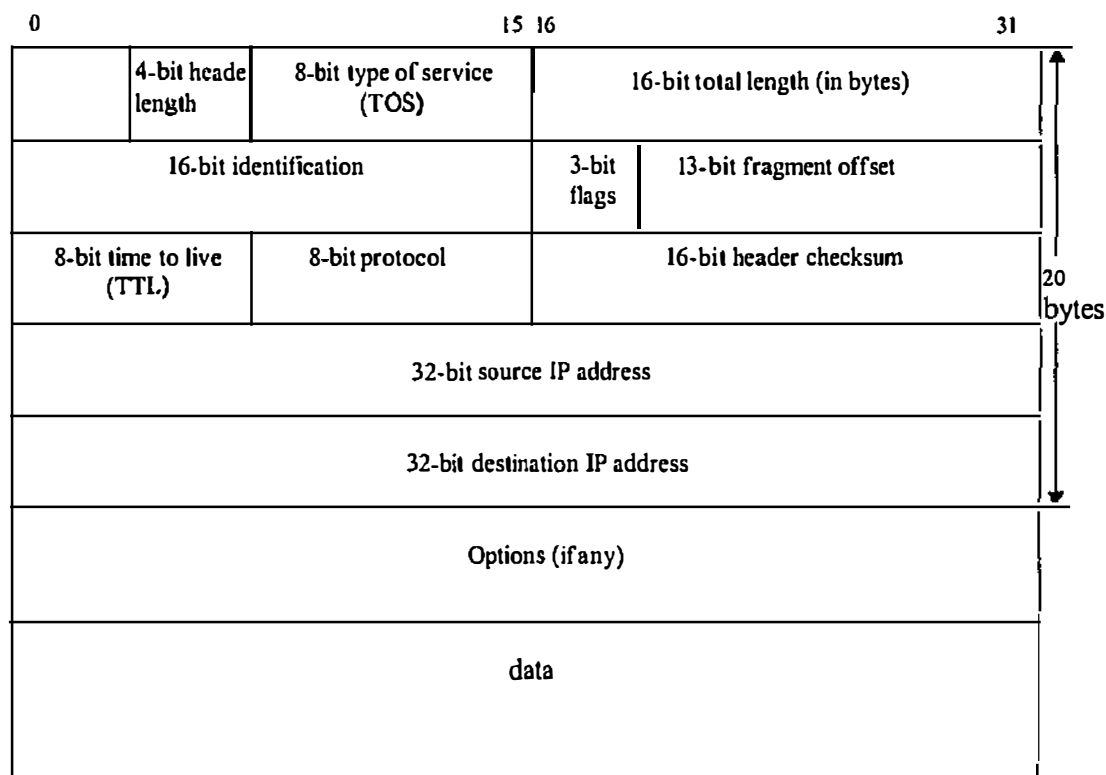
2.2.3 IP: Internet Protocol

IP is the protocol that hides the underlying physical network by creating a virtual network view. It is an unreliable, best-effort, and connectionless packet delivery protocol. There is no guarantee that an IP packet successfully gets to its destination. IP is considered as best effort service because when something goes wrong, IP has a simple error handling algorithm which throws away the packet and try to send an ICMP message back to source (Steven, 1994).

IP does not maintain any state information about successive packets. Each packet is handled independently from all other packets. The delivery of packets may not be in order.

2.2.3.1 IP Header

The IP packet header consists of 20 bytes of data. An option exists within the header which allows further optional bytes to be added, but this is not normally used. The full header is shown below:



(Steven, 1994, p34)

Figure 2.5: IP Header Structure

The following paragraph explains the structure of IP Header as explained by Steven (1994). The *header length* is the number of 32 bit words in the header, including any options. Since it is a 4-bit field, it limits the header to 60 bytes. The *type of service* field (TOS) is composed of a 3-bit precedence field, 4 TOS bits are unused bit that must be 0. The 4 TOS bits are: minimize delay, maximize throughput, maximize reliability and minimize monetary cost. Only 1 of these 4 bits can be turned on. If all 4 bits are 0 it implies normal service.

The *total length* field is the total length of the IP packet in bytes. Using this field and the header length field, we know where the data portion of the IP packet starts and its length. Since this is a 16-bit field, the maximum size of an IP packet is 65535 bytes. This field also changes when the packet is fragmented.

The *identification* field uniquely identifies each packet sent by a host. It normally increments by one each time a packet is sent. It is also used during reassembly of

fragmented packets. The *flag* field is composed of a sequence of 3 flags used to control whether routers are allowed to fragment a packet and to indicate the parts of a packet to the receiver. The 13-bit *fragment offset* consist of a byte count from the start of the original sent packet, set by any router which performs IP router fragmentation (Steven, 1994; Comer, 1999; Parker, 2000).

The *time-to-live* field, or TTL, sets an upper limit on the number of routers through which a packet can pass. It limits the lifetime of the packet. It is initialised by the sender to some value and decremented by one by every router that handles the packet. When this field reaches 0, the packet is dropped and the sender is notified with an ICMP message.

The *protocol* identifies which protocol gave the data for IP to send.

The *header checksum* is calculated over the IP header only. It does not cover any data that follows the header. A 2's complement checksum inserted by the sender and updated whenever the packet header is modified by a router. Used to detect processing errors introduced into the packet inside a router or bridge where the packet is not protected by a link layer cyclic redundancy check. Packets with an invalid checksum are discarded by all nodes in an IP network (Steven, 1994; Comer, 1999)

Every IP packet contains the *source IP address* and the *destination IP address*.

2.2.3.2 IP Routing

An important function of the IP layer is *IP routing*. An internet is composed of multiple physical networks interconnected by computers called routers. Each router has direct connections to two or more networks. A host computer usually connects directly to one physical network. The router only has information about four kinds of destinations:

- Hosts that are directly attached to one of the physical networks to which the router is attached.

- Hosts or networks for which the router has been given explicit definitions.
- Hosts or networks for which the router has received an ICMP redirect message.
- A default for all other destinations.

(Steven, 1994)

If the destination is directly connected to the host or on a shared network then the IP packet is sent directly to the destination. Otherwise the host sends the packet to a default router to deliver the packet to the destination.

Direct and Indirect Routing

IP routing can be divided into two forms: **Direct routing** and **Indirect routing** (Rodriguez, 2001). In direct routing, packets are transmitted between two machines which are on a single physical network and also do not involve routers. The sender encapsulates the packet in a physical frame, binds the destination IP address to a physical hardware address, and sends the resulting frame directly to the destination. Indirect routing occurs when the destination host is not connected to a network directly attached to the source host.

Indirect routing is more difficult than direct routing because the sender must identify a router to which the packet can be sent. When one host tries to send an IP packet to other host, it encapsulates the packet and sends it to the nearest router. Once reaching to the router, the IP software selects the next router along the path towards the destination. The packet is again passed on to the next physical network and the process continues until the packet reaches its destination.

IP routing table

The IP layer has a routing table in memory that it searches each time it receives a packet to send. It consists of a list of local networks and indirect routes. Three types of mappings are found in this table (Rodriguez, 2001):

1. The direct routes describing locally attached networks.
2. The indirect routes describing networks reachable via one or more gateways.

3. The default route which contains the (direct or indirect) route used when the destination IP network is not found in the mappings of type 1 and 2 above.

When a packet is received from a network interface, IP first checks if the destination address is one of its own IP addresses. If so, the packet is delivered to the destination on the local network. If the packet's destination address is not of this IP layer, it is forwarded to the destination IP network via one or more gateways else the packet is forwarded to the default route. If none of the above steps works, the packet is undeliverable. At this stage, a "host unreachable" or "network unreachable" message is returned to the source address from where the packet was originated.

2.2.4 TCP: Transmission Control Protocol

TCP is a communication protocol which provides a connection oriented, reliable, byte stream service. The two machines need to make a TCP connection before exchanging any data between them. TCP provides reliability by doing the following:

- TCP breaks down big packets into small size packets called segments.
- After sending the segment across the network, it waits for an acknowledgement. If an acknowledgement is not received in time, the segment is retransmitted.
- Acknowledges the receipt of packet from other end.
- Maintains a checksum on its header and data to detect any modification of the data in transit.
- IP packets may not arrive in sequence, so TCP re-sequence them and pass to the recipient in correct order.
- IP packets may be duplicated, so it discards duplicate data.
- TCP also provides flow control. Each end of TCP connection has a finite amount of buffer space.

(Steven, 1994)

2.2.4.1 TCP Header

The TCP packet header consists of 20 bytes of data. An option exists within the header which allows further optional bytes to be added, but this is not normally used. The full header is shown below:

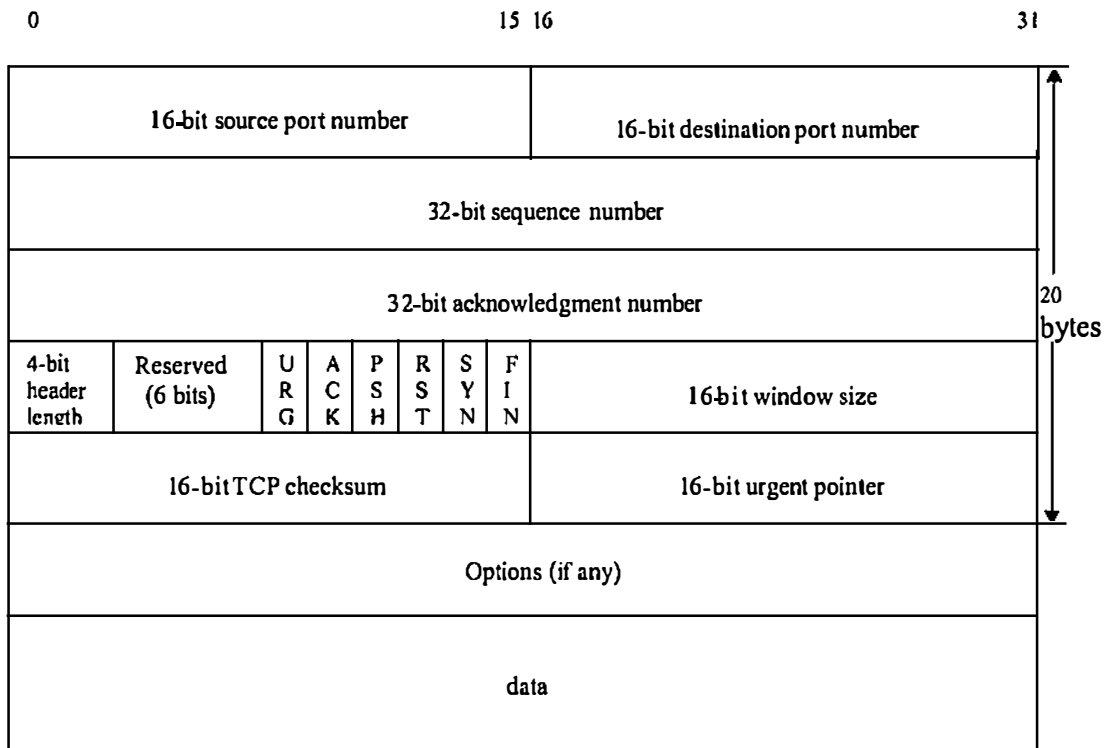


Figure 2.6: TCP Header

(Steven, 1994, 225)

Each TCP segment contains the source and destination *port number* to identify the sending and receiving application. These two values along with the source and destination IP address in the IP header uniquely identify each connection. The *sequence number* identifies the byte in the stream of data from the sending TCP to the receiving TCP that the first byte of data in this segment represents. Since every byte that is exchanged is numbered, the *acknowledgement number* contains the next sequence number that the sender of the acknowledgement expects to receive. This is therefore, the sequence number plus 1 of the last successfully received byte of data. The *header length* gives the length of the header in 32-bit words.

There are six flag bits in the TCP header. One or more can be turned on at the same time. These flags are briefly described below (Steven, 1994):

- URG The *urgent pointer* is valid
- ACK The *acknowledgement number* is valid
- PSH The receiver should pass this data to the application as soon as possible.
- RST Reset the connection
- SYN Synchronize sequence numbers to initiate a connection.
- FIN The sender is finished sending data.

The 16-bit window size provides flow control in TCP. The *checksum* covers the TCP segment. It is a compulsory field that must be calculated and stored by the sender, and then verified by the receiver. The *urgent pointer* is valid only if the URG flag is set. This pointer is a positive offset that must be added to the sequence number field of the segment to yield the sequence number of the last byte of urgent data. TCP's urgent mode is a way for the sender to transmit emergency data to the other end.

The most common option field is the maximum segment size option, called the MSS. Each end of a connection normally specifies this option on the first segment exchanged. It specifies the maximum sized segment that the sender wants to receive (*ibid*, 1994).

2.2.4.2 Connection and Termination

A successful TCP connection involves a three-way handshake. The figure 2.7 shows the proceeding of handshake.

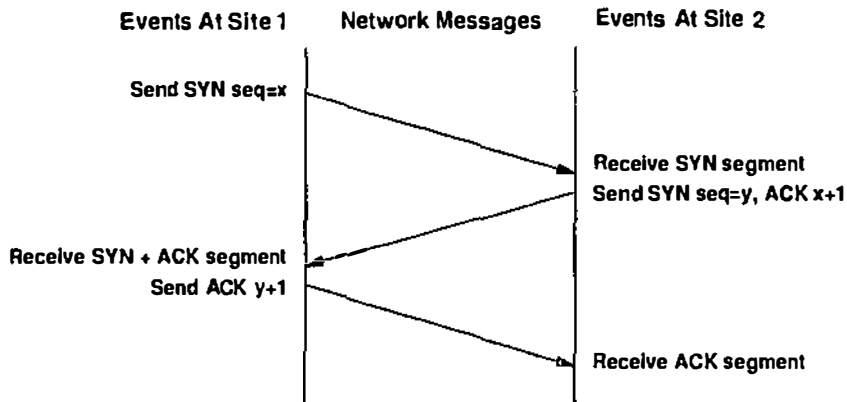


Figure 2.7: TCP Connection

(Comer, 1995, p216)

To establish a TCP connection:

- The requesting host sends a SYN segment specifying the port number of the server that the client wants to connect to, and the client's initial sequence number.
- The server responds with its own SYN segment containing the server's initial sequence number. The server also acknowledges the client's SYN by sending its acknowledgement, ACK, which is clients initial sequence number plus one.
- The client must acknowledge this SYN from the server by sending its ACK, which is server's initial sequence number plus one.

(*ibid*, 1995)

This 3-way handshake completes the connection establishment.

It is a four segment process to terminate an established TCP connection. Since TCP is a full duplex (that is, data can be flowing in each direction independently of the other direction.), each direction must be shut down independently. The termination process of a TCP connection between server and client is shown below:

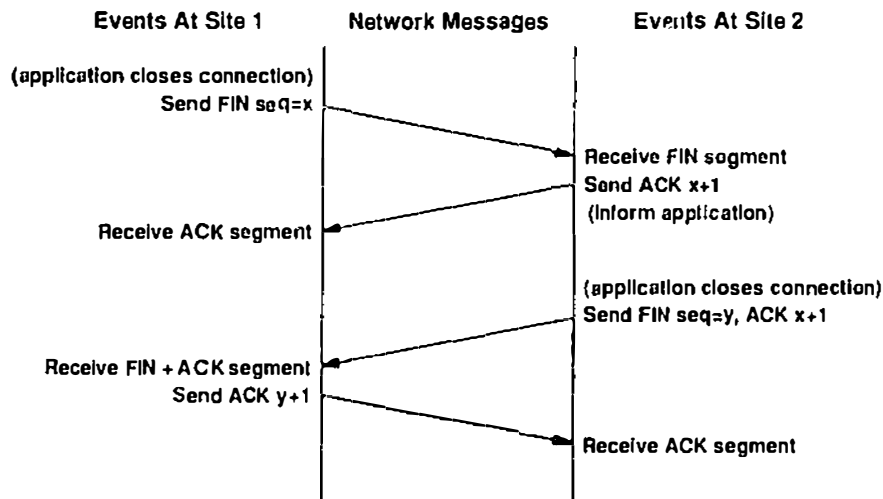


Figure 2.8: TCP Connection Termination Process

(Comer, 1995, p218)

Either end can send a FIN when they are done sending data. When TCP receives a FIN, it must notify the application that the other end has terminated that direction of data flow. Sending a FIN normally means that application issuing a close. The receipt of a FIN means that there will be no longer flow of data in that direction. A TCP can still send data after receiving a FIN. When the server receives the FIN it sends back an ACK of the received sequence number plus one. At this stage, server's TCP also delivers an end-of-file to the application. The server then closes its connection, causing its TCP to send a FIN which the client TCP must ACK by incrementing the received sequence number by one (*ibid*, 1995).

2.2.4.3 TCP State Machine

TCP finite state machine is a theoretical model which explains the operation of TCP at its best. The figure below shows the TCP finite state machine, with circles representing states and arrows representing transitions between them. The label on each transition shows what TCP receives to cause the transition and what it sends in response.

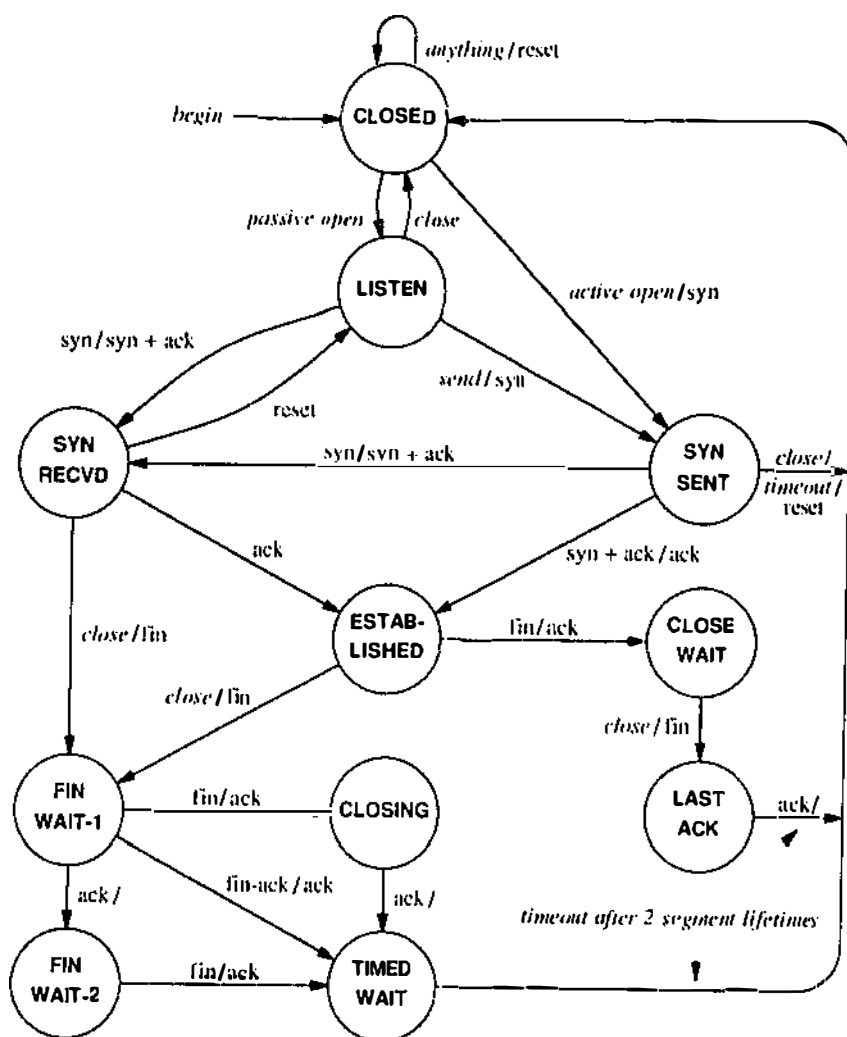


Figure 2.9: TCP State Machine

(Comer, 1995, p220)

The TCP software at each endpoint begins in the *CLOSED* state. Application programs can either have an *active open* (to initiate a connection) or a *passive open* (to wait for the connection from other machine). In an *active open* situation, the transition changes from *CLOSED* state to *SYN SENT* state with the emission of *SYN* segment. After receiving a *SYN* segment, the other end of the connection a segment with both *SYN* and *ACK* and moves the transition to *ESTABLISHED* state and begins data transfer (*ibid*, 1995).

The *TIMED WAIT* state in the above figure reveals how TCP handles the problems incurred with unreliable delivery. TCP keeps track of maximum segment lifetime, the

maximum time an old segment can remain alive in the network. When a request of closing a connection is received, TCP moves to the *TIMED WAIT* state in order to avoid any interference between previous connection and the current one. TCP rejects any duplicate segments received during the time out interval but however accepts the valid segments and restart the timer. The timer allows TCP to distinguish old connections from new ones; it prevents TCP from responding with *RST* (reset) if the other end retransmits a *FIN* request (*ibid 1995*).

2.2.5 Security Issues with TCP/IP

This section gives an overview of some of the most common attacks on TCP stack level and also their defences (Bellovin, n.d; NORMAN, 2003).

2.2.5.1 TCP Sequence Number Prediction

A TCP connection establishment requires a 3-way handshake. As described in section 4.4.2, the client transmits an initial sequence number *ISN C*, the server acknowledges it and sends its own sequence number *ISN S* and the client acknowledges. Once this 3 way handshake completes, the data transmission begins. This exchange may be shown schematically as follows:

```
C -> S:SYN(ISN C)
S -> C:SYN(ISN S), ACK(ISN C)
C -> S:ACK(ISN S)
C -> S:data
And / or
S -> C:data
```

Many TCP connections use predictable *ISNs* for the above purpose. As a result, the malicious attacker could guess the server's *ISN* used during the connection establishment procedure and trick the server into believing that itself is the client or the trusted machine. *ISNs* are easy to guess because they are incremented by a

constant amount each second or half that amount each time a connection is initiated. This may be shown schematically as follows:

$X \rightarrow S: SYN(ISN\ X), SRC = T$
 $S \rightarrow T: SYN(ISN\ S), ACK(ISN\ X)$
 $X \rightarrow S: ACK(ISN\ S), SRC = T$
 $X \rightarrow S: ACK(ISN\ S), SRC = T, \text{ nasty - data}$

Even the message $S \rightarrow T$ does not go to X , X was able to know its contents, and hence could send the data.

Defence

According to Bellovin (n.d), the best way to prevent such attacks is to make the ISNs more difficult to predict. Making use of cryptographic algorithms such as DES for random ISN generation can make the ISNs hard to guess.

2.2.5.2 ICMP attack

The Internet Control Message Protocol (ICMP) is the basic management tool of the TCP/IP protocol suite. It is an error reporting protocol. There are various security holes in ICMP which may be exploited (Low, 2001).

Firstly, is the ICMP *Redirect* Message which is used to advise hosts for better routes on a network. The redirect message can be used to create false routes to a trusted host through a compromised router or gateway. Suppose an intruder setup a false route to a trusted host from a compromised gateway. The following sequence of events may occur. Send a false TCP open packet to the target host, claiming to be from a trusted host T. The target will respond with its own open packet, routing it through the secure primary gateway. While this packet is on its way to trusted host, a false redirect may be sent claiming to be from the primary gateway but referring to the false connection. Since the packet will be assumed to be coming from a legitimate source, the routing change will be readily accepted as hosts do not perform enough validity checks on

ICMP redirect messages. As a result, the intruder may be able to establish a connection and spoof the trusted host T (Bellovin, n.d; Low, 2001).

ICMP may also be used to carry out Denial of Service (DoS) attacks by resetting connections using error messages such as *Destination Unreachable* or *Time-to-Live (TTL) exceeded* (Low, 2001).

Defences

Strict validity checks need to be done on ICMP packets thereby preventing forging of connections. The ICMP redirect messages can also be restricted to a connection rather than to the global routing table.

2.2.5.3 SMURF Attack

The Smurf attack is not intended to halt a computer but a network of computers. This is a type of network security breach in which a network connected to the Internet is swamped with replies to ICMP echo (PING) requests.

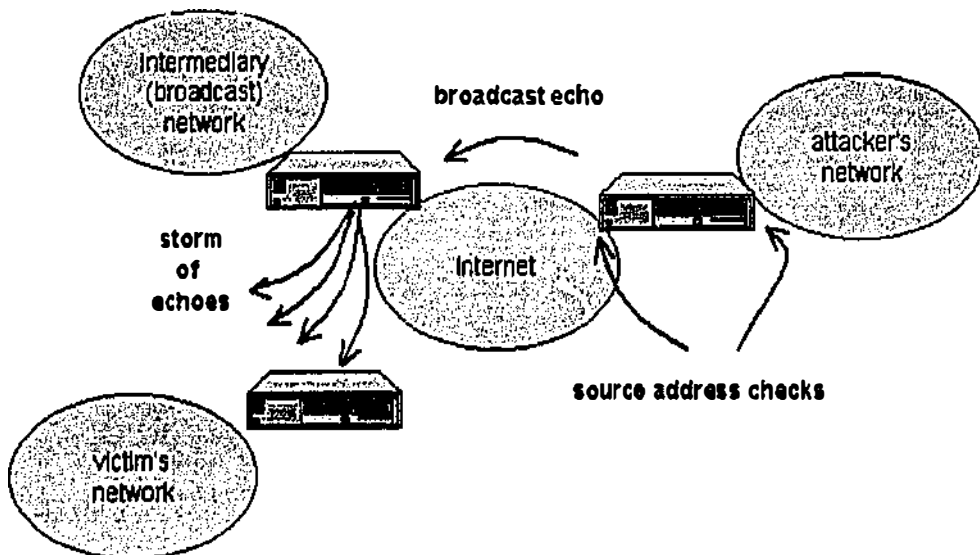


Figure 2.10: Smurf Attack

(Low, 2001)

A smurf attack is illustrated below (Low, 2001):

Step 1. Attacker finds some intermediary network that will respond to the network's broadcast address.

Step 2. Attacker spoofs the IP address of the victim host and sends a great number of ICMP echo request packets to the broadcast address of the above intermediary networks.

Step 3. Now all the hosts on that network will respond to that ICMP echo request with a corresponding ICMP reply request back to the spoofed IP address (the victim).

Step 4. This will send a large amount of ICMP echo replies to the victim and its network thus causing network degradation or a total denial of service.

Defences

Following defensive measures can be used to protect a network from smurf attacks (Hirani, et. al, n.d):

- Disabling broadcast messages
- Upstream firewalls should be configured to either filter ICMP echo responses or limit echo traffic.
- In order for these attacks to take place a router must be configured to allow packets to exit the network with a source address that does not originate from the internal address. It is possible to configure your router to enable network egress filtering.

Network egressing is when a router is configured to filter out packets that do not originate from the internal network.

ISPs can also employ network ingress filtering to drop packets that do not originate from a known range of IP address, which is effective not only for preventing local origination of SMURF attacks but also for ease in tracking attacks.

2.2.6 TCP/IP Fingerprinting

To attack any system, it is very important to gather information about the target machine. There can be various ways which can be used to gather information about the target machine. One of the most important pieces of information that an attacker or hacker would like to know about the target machine is the type and version of Operating System (OS) running on it. Knowing the version and type of OS can help an attacker to gather information regarding the vulnerabilities related to that OS.

There are various tools available on the Internet which can help in determining the operating system of a host by examining details in the way the TCP/IP stack was implemented within that operating system. This method is called TCP/IP fingerprinting (Fyodor, 1998).

2.2.6.1 Classical Methods

The easiest method to identify the operating system is to contact it by its open port. Most telnet, FTP and web servers will not only identify themselves but will also give information about the operating system running on them. For example (*ibid*, 1998):

```
root%> telnet hpux.u-aizu.ac.jp
Trying 163.143.103.12 ...
Connected to hpux.u-aizu.ac.jp
Escape character is '^]'

HP-UX hpux B.10.01 A 9000/715 (ttyp2)

Login:
```

In the above example, the machine itself explains what it is running. Such methods are sometime enough to determine information about the operating system running on the target machines.

Another classic technique for identifying the operating systems includes port sweeps and grabbing email headers. Port sweeps identify open ports, and on a system that has not been hardened, this list of open ports is often enough to identify most operating systems. E-mail headers can identify mail user agents, the user's operating system, the mail server and sometimes even the firewall that the mail passed through.

2.2.6.2 TCP/IP Fingerprinting with Nmap v 3.0

Utilities such as Nmap are designed to allow system administrators and curious individuals to scan large networks to determine which hosts are up and what services they are offering. Nmap supports a large number of scanning techniques such as: UDP, TCP connect(), TCP SYN (half open), ftp proxy (bounce attack), Reverse-ident, ICMP (ping sweep), FIN, ACK sweep, SYN sweep IP protocol and Null scan. These techniques are briefly described below (Fyodor, 1998; Glaser, 2000):

- ❖ *UDP and TCP connect()*: This type of scan method involves opening a full connection to a remote host using a typical three-way TCP/IP handshake. For example,

```
Client -> SYN
Server -> SYN|ACK
Client -> ACK
```

The above example shows a port answering our initial connection request with a SYN|ACK. Therefore the port is in the open state. Once the full handshake is taken place, the connection will be terminated by the client. If the port is in the closed state then the response will be:

```
Client -> SYN
Server -> RST|ACK
Client -> RST
```

The RST|ACK flags returned by the server shows that the port is not in the listening state thus is closed.

- ❖ *Half open scan*: The term 'half open' applies to the way the client terminates the connection before the three-way handshake is completed. This scan method often

goes unlogged by IDS and also returns fairly positive results (Fyodor, 1998; Glaser, 2000).

- ❖ *FTP server bounce attack*: Under this attack, a host is able to determine the status of a port by issuing an IP and port as arbitrary parameters to connect to. If a connection is established as a means of active data transfer processing, the client knows a port is open otherwise a 425 error message will be generated (Fyodor, 1998).
- ❖ *Reverse-Ident*: This technique involves issuing a response to the idnet/auth server, usually port 113 to query the service for the owner of the running process. Ident could release miscellaneous private information such as(Ibid, 1998):
 - user info
 - entities
 - objects
 - processes
- ❖ *ICMP ping sweep*: An ICMP ping sweep is a basic network scanning technique used to determine which of a range of IP addresses map to live hosts (computers). Whereas a single ping will tell you whether one specified host computer exists on the network, a ping sweep consists of ICMP (Internet Control Message Protocol) ECHO requests sent to multiple hosts. If a given address is live, it will return an ICMP ECHO reply (Anonymous, n.d).
- ❖ *FIN scanning*: The FIN scan method uses inverse mapping to discover closed ports. Once a FIN flagged packet is sent, a closed port will respond with an RST bit. Open ports will not send a packet back (Fyodor, 1998).
- ❖ *ACK scanning*: This technique is used to identify open TCP ports by sending probe TCP packets with the flag ACK set, and then analysing the header information of the RST packets received from the target host. This technique exploits vulnerabilities within the BSD derived TCP/IP stack, and is therefore

only effective against certain operating systems and platforms. There are two main ACK scanning methods (*ibid*, 1998):

- Assessment of the TTL field
- Analysing the Window field

❖ *Null Scan*: The Null scan unsets ALL flags available in the TCP header. ACK, FIN, RST, SYN, URG, PSH all become unassigned. If the port is open then the incoming packet is dropped otherwise an RST packet will be returned if a closed port has been reached.

Nmap also offers a number of advanced features such as remote OS detection via TCP/IP fingerprinting, stealth scanning, dynamic delay and retransmission calculations, parallel scanning, detection of down hosts via parallel pings, decoy scanning, port filtering detection, direct (non-port mapper) RPC scanning, fragmentation scanning and flexible target and port specification.

Every operating system has different TCP/IP stack's in compare to other operating systems; therefore each one has a different response in a TCP/IP conversation. This is because each TCP/IP stack has different sequencing of the packet and will have different flags set due to implementation. Nmap interrogates the target machine's TCP/IP stack by sending it eight different packets and observing the response. Nmap interrogates the target machine's TCP/IP stack by sending it eight different packets and observing the response. The packets are

“Tseq is the TCP sequenceability test
T1 is a SYN packet with a bunch of TCP options to open port
T2 is a NULL packet w/options to open port
T3 is a SYN|FIN|URG|PSH packet w/options to open port
T4 is an ACK to open port w/options
T5 is a SYN to closed port w/options
T6 is an ACK to closed port w/options
T7 is a FIN|PSH|URG to a closed port w/options
PU is a UDP packet to a closed port” (Fyodor, 1998)

The test is specifically crafted to put the target machine in a position where there is a high probability that two things will happen. Firstly that the target operating system's TCP/IP stack will respond unique in comparison to another operating system's

TCP/IP stack. Secondly that the target operating system's TCP/IP stack will respond in a consistent manner.

The granularity of this is reasonably high for example here are 2 fingerprints both of NT4 machines take from the NMAP fingerprint files of varying Service Pack Levels

Fingerprint Windows NT 4 SP3

```
TSeq(Class=TD|RI%gcd=<18%SI=<2A00DA&>6B73)
T1(DF=Y%W=7FFF|2017%ACK=S++%Flags=AS%Ops=M|MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=7FFF|2017%ACK=S++|O%Flags=AS|A%Ops=M|NNT)
T4(DF=N%W=0%ACK=O|S%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O|S++%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E
)
```

Fingerprint Windows NT 4.0 SP 6a + botfixes

```
TSeq(Class=RI%gcd=<6%SI=<40132&>290%IPID=BI|RPI%TS=U)
T1(DF=Y%W=2017%ACK=S++%Flags=AS%Ops=M)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=2017%ACK=S++%Flags=AS%Ops=M)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=C00|800%ACK=S++%Flags=AR%Ops=WNMETL)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=WNMETL)
T7(DF=N%W=0%ACK=O%Flags=R%Ops=WNMETL)
PU(Resp=N|Y)
```

This attack intelligence allows an attacker to quickly narrow the choices of attack dependent upon the information that NMAP reports back. This attack intelligence gives an attacker several advantages. Firstly, they can use a few attacks or probes on the system as they now have a reasonable idea of installed operating system. Secondly, as a result of having knowledge of the operating system by using a few attacks may reduce the probability of detection by defensive systems.

2.3 Honeypots

This section of the chapter initially defines what are honeypots and honeynet. Later it outlines various advantages and disadvantages of honeypots. Section 2.3.3 outlines various types of honeypots available. Section 2.3.4 explains the architecture of honeyd - an open source honeypot. Section 2.3.5 discusses the various methods of deploying the honeypots.

2.3.1 What are Honeypots and Honeynet?

According to Spitzner (2003), “*A honeypot is a security resource whose value lies in being probed, attacked or compromised.*” These systems are specially designed to attract attackers. Their purpose is to capture all activities and any changes made in the system by attackers’ who would have gained unauthorised access. Honeypots are seen as research tools which allow learning from attackers’ skills and knowledge. They can be used for attack intelligence gathering about various attacks and exploits explored by attackers. Honeypots can be used to achieve various goals. They can be used to deter attacks, to detect attacks and also to capture and analyse attacks. They can be used as a learning tool to learn and understand about various attacks and attacking tools used by attackers. (Brenton, n.d; Klug, 2000; Spitzner, 2002). Various network services are emulated on the honeypot network which is attractive targets for the attackers. Services like *http (port 80)*, provides access to the web-based systems such as web servers which may be the prime target for attacking any network. If the web-server is compromised, the attacker is capable of posting their own web pages or any web related exploits on the network such as any back doors or Trojans. Also services like *ftp (port 21)*, if compromised may allow the attackers to upload their own malware programs and utilities which may be used for compromising the whole network.

Example of Honeypot

To understand the concept of a honeypot and how they work let us consider an example. Assume that an organisation is interested in upgrading their old servers such as mail server, web servers etc. which have been currently in use. Before upgrading, these main production systems were used for providing services to the organisation.

These old servers can be readily utilised as a honeypot servers which can be left within the organisation's DMZ (Demilitarised Zone) (Figure 2.11, p50). Any connections made to the old server machines, now honeypots, could be reasonably assumed to be suspicious in nature. This possibly means that someone is trying to probe or attack the servers or the organisation as a whole.

If it is found that connections are from the honeypot to the main internal network that would indicate the honeypot is compromised.

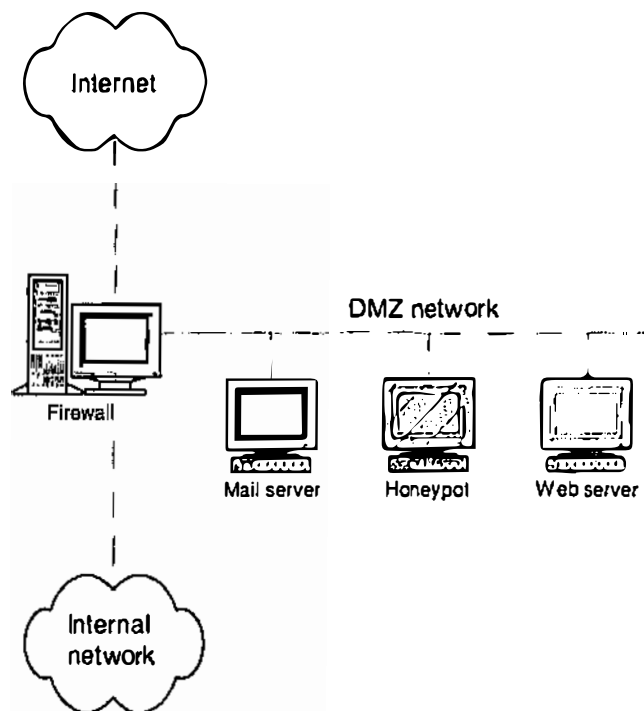


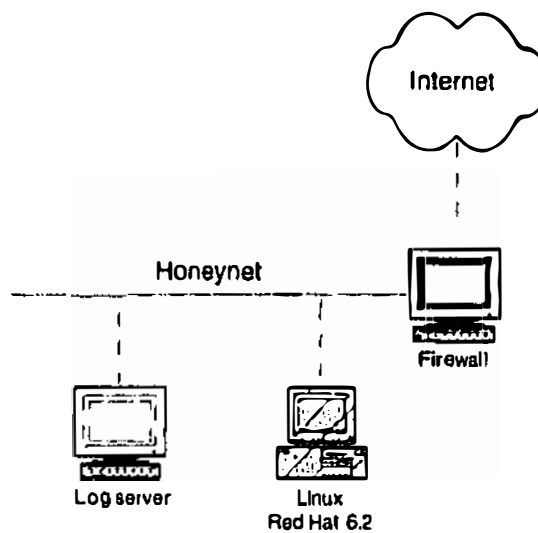
Figure 2.11: Network diagram of a honeypot deployed on a DMZ to detect attacks

(Spitzner, 2003, p63)

According to Spitzner (2003), a *honeynet* is a high interaction honeypot designed particularly for research, to gather information on the attackers. It is a network of multiple honeypots which are probed and attacked by attackers. This could be a complete separate network segment within the network architecture of any organisation. In a honeynet, there can be various types of systems such as Windows, Linux, etc... which can be configured as honeypots. This creates a network environment that has a more realistic approach to it for the attacker. All systems

placed within the Honeynet are the mirror images of the standard systems placed within the internal network of any organisation (Anonymous, 2002a). Therefore, honeynet are high interaction honeypots designed to gather information about attackers.

According to the Deception planning process, as discussed in section 2.2 (p.12), the objective of deploying the honeynet is to create a deceptive network to deceive the attackers and deterring them from causing harm to the real production systems. Another important objective of the honeynet is to gather attack intelligence about various attacks carried out by attackers (targets). To attract these targets, a deceptive network is created with various hosts with number of operating systems and services configured over them. These systems are configured with some known vulnerabilities to attract the attackers.



(Spitzner, 2003, p266)

Figure 2.12: The Honeynet

2.3.2 Value of Honeypots

Honeypots can be of great value for an organisation as they may help in protecting an organisation by preventing, detecting or responding to an attack. Honeypots can be used for prevention purposes in various ways. They can be used against the automated attacks such as auto-rooter or worms. These are programs or algorithms that search

for vulnerable systems over the computer network and replicate it and usually perform malicious actions, such as using up the computer's resources and possibly shutting the system down (Anonymous, n.d(c)). Honeypots can be of help in defending against such attacks by slowing down attackers' access to the systems, i.e. by taking extra time in processing attackers' requests or also by completely shutting down the system. The idea here is to distract or fool the attacker to make him or her wastes their resources and time while interacting with honeypots. This provides deterrence which may prevent attacks on production systems.

The second level of security is detection. The purpose of detection is to identify failures and unauthorised activities within the honeypot. If someone has to secure their house, the detection would be possible by installing burglar alarms or motion detectors. Similarly there are various technologies which can be used for detection in a network environment. For example, Network Intrusion Detection Systems which are developed to monitor networks and identify malicious activity. There are also programs that monitor system logs and identify any unauthorised activity. But these technologies have their own limitations. They generate large amounts of data such as false positives and false negatives. Honeypots can be very efficient in overcoming these limitations of the traditional method of detections. They can reduce false positives by capturing small data sets of high value which may contain new attacks or rootkits or encrypted packets. Since honeypots are meant to be probed and attacked, so any traffic is suspicious in nature therefore there are very less chances of having any false positives. Honeypots are also very effective in capturing new exploits as they can detect the attacks according to system activity, not signatures. This helps in reducing the false negatives by honeypots (Anonymous, 2002(a); Spitzner, 2003).

Once the organisation has detected any attack, now they need to respond to it. Before responding to any attack, it is essential for any organisation to collect all the evidence related to the attack such as who the attacker is, how they entered into the network, what damage did they caused. It is a challenge to collect all the information related to the incident from the compromised systems (Spitzner, 2003). For example, production systems are very critical for any organisation such as their mail server. It will not be possible for them to take the server offline for conducting forensic analysis. When the system is brought offline, a major problem of data pollution arises for organisations.

There are various activities which occur in the system. For example, files written to the hard drive, processes execution and termination, user login details etc., therefore it may not be easy to determine what the everyday activity was and what the attacker was trying to implement or access in the systems. Honeypots can be of great value in such situations. They can be easily taken offline without hindering any everyday activity and can be used for complete forensic analysis. Honeypots can be setup to only collect unauthorised or malicious activity so this makes data analysis much easier. In order to respond to an attacker's activity, it is essential to understand about what they did, how they did and what damage was done.

Any traffic, inbound or outbound, to honeypot is considered to be suspicious by nature. Because of the design assumptions of honeypots they have certain inherent advantages and disadvantages (Anonymous, 2000; Sink, 2001; Spitzner, 2002) which are outlined below:

Advantages

- ❖ **Data Collection:** Honeypots collect normally data of very high value. Honeypots can collect small data sets. These data sets may be tools, rootkits or scripts used by the attackers. Such forensic evidence provides a lot of useful information on attacker's movement within the system and their (attackers) interaction with the system in a consistent format from a variety of sources. Such data is of high value as it only provides information about the malicious attacks. (Spitzner, 2002).
- ❖ Allows security researchers and security practitioners to study exactly what attackers are doing to compromise or probe systems. This is attempted to be done without exposing systems and networks to the additional risk that results from compromised systems (Anonymous, 2000).
- ❖ Honeypots provide delay (Sink, 2001). As a result of ruse, camouflage or false information honeypots consume attackers' time and resources. This helps in delaying them in attacking the real servers in the network. Methods such as network delays and response time while establishing any network connection or causing time delays while an attacker is trying to download may sometime frustrate an attacker. This may provide enough time to an organisation to detect such activity and respond to them.

Disadvantages

- ❖ They are worthless if no one attacks them. Honeypots are useful when someone attacks or probe them. They may not be of any use if nobody interacts with these systems (Spitzner, 2002).
- ❖ Honeypots can be used as a launching platform to attack other machines on different network and hence need sophisticated protection mechanisms and controls (Sink, 2001; Spitzner, 2002).

2.3.3 Types of Honeypots

There are various types of honeypots available each with their own distinct features. Some of the honeypots are discussed below (Spitzner, 2002; Scottberg, 2002):

2.3.3.1 BackOfficer Friendly (BOF)

BOF is a very simple honeypot developed by Marcus Ranum and his colleagues at NFR (Network Flight Recorder). It is a low interaction honeypot. BOF is a program that runs on most Windows based operating system. It can emulate basic services, such as ftp, http, telnet, mail or BackOrifice. When someone tries to connect to BOF, it can log the attempt and an alert is generated. It also provides the feature of sending “pre-defined replies” as well to attackers to keep them busy. It can only monitor a limited number of ports, but these ports represent the most commonly scanned and targeted services. None of the services emulated by BOF are based applications or version, only the functionality of the service is emulated. For example, it can emulate a web server but it does not emulate specific web server like Apache or Microsoft IIS. Therefore, it is difficult to use BOF as a prevention tool as it offers very little interaction with the attacker.

2.3.3.2 Specter

Specter is a commercial honeypot created and supported by NetSec, a network security company based in Switzerland. It is similar to BOF but has greater functionality. It is a Windows based honeypot and can emulate various services an attacker can interact with. Specter does not have any real applications installed on it

instead some limited emulated functionality so there is not much for attackers to do on the system. Specter is equipped with seven fully emulated services, six traps and one customizable trap. This provides enough flexibility to detect attacks on 13 predefined ports and 1 configurable port. One of the unique features of Specter is that it not only can emulate services but also vulnerabilities as well. An attacker may launch an attack on Specter and may believe it to be successful but it may not be (Spitzner, 2003). It supports various logging and alerting mechanisms, which in turn helps in learning the attacker's moves. Specter also allows information gathering but this is at a very low level. Some of the information gathering is relatively passive, such as Whois or DNS lookup, which may just provide information about the attackers location and system but it does not capture or provide information regarding any tools or scripts used by the attacker.

Specter emulates operating systems only at an application level; there is no emulation with the underlying IP stack. Therefore, the emulated operating system may not match with the IP stack may identify the true purpose of the honeypot (Anonymous, n.d(e)).

2.3.3.3 Homemade Honeypots

These honeypots are mainly used to capture malicious codes such as worms or viruses and scanning activities. Homemade honeypots may have variety of possible uses depending on the developer's requirement and creativity. They can be used for just detecting port scans or any specific probe thus providing a low level of interaction to the attackers. They can also be designed and developed to emulate a complete operating system and allowing attackers to probe and execute their programs. This will provide greater level of interaction. Some common examples of homemade honeypots are (Spitzner, 2003):

- Windows Inetd emulator for Windows NT and Win2000.
- Sendmail Honeypots, used to identify sendmail spammers.

2.3.3.4 Honeyd

Honeyd was created and developed by Nicks Provos of University of Michigan in April 2002. It is an open source honeypot which runs on UNIX platform and can emulate over 400 operating systems or network devices. It can potentially simulate

networks of thousands of computers. Unlike Specter, it emulates the operating system at IP stack level as well. This means that when an attacker tries to probe or scan the network or honeypot using network scanning tools, such as Nmap, both the service and IP stack behave as the emulated operating system.

Honeyd is primarily used for detecting attacks. It works by monitoring IP addresses that are unused, that have no device assigned to them. Services emulated using Honeyd can be simple scripts to full interaction on the native operating system via port redirection that react to predetermined actions. Honeyd can not only dynamically interact with attackers, but it can detect activity on any port. It detects and logs connections made to any port, regardless if there is a service listening. It is capable of emulating different operating systems or network devices at the same time. The combined capabilities of assuming the identity of non-existent systems, and the ability to detect activity on any port, give Honeyd incredible value as a tool to detect unauthorised activity.

2.3.3.5 Mantrap

It is a commercial honeypot produced by Resource Mantrap. Instead of emulating services, it creates 4 sub-systems called 'jails'. These 'jails' can be configured by system administrator as they want. It is much more flexible in compare to other honeypots. It not only detects attacks, port scans but also helps in capturing rootkits, IRC chat sessions as well. As it is a commercial honeypot and depends on vendor specification, it has a limited use. At present, it is available on Solaris operating system only.

The above mentioned types of honeypots can be summarised in the table below (Adapted from Baumann, 2002):

	ManTrap	Specter	BOF	Honeyd	Homemade
Degree of Involvement	High	Low	Low	Middle	Low– high
Expandable	✓	-	-	✓	✓
Open Source	-	-	-	✓	N/A
Freely available	-	-	✓	✓	✓
Log file support	✓	✓	-	✓	N/A
Notification mechanisms	✓	✓	-	✓	N/A
Supported services	Unlimited	13	7	Over 400	N/A
Configuration complexity	High	Low	Low	Medium	N/A
GUI for Control and Logging	✓	✓	✓	-	N/A

Table 2.1: Summary of Different types of Honeypots

On the basis of the above discussion about various honeypots and their features and performance, a following graph can be plotted regarding their performance and complexity in use in a network environment (Developed by researcher):

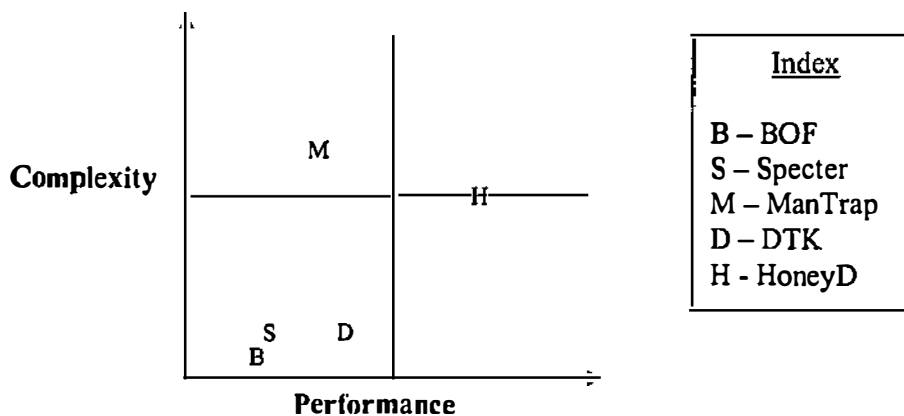


Figure 2.13: Complexity vs. Performance Graph of various Honeypots

From the above graph 2.13, it can be interpreted that Honeyd is the only honeypot which provide high performance with higher complexity. Therefore, it may be noted that Honeyd may suit best to the researcher's requirement. The detail concept of honeyd is explained in the below section.

2.3.4 Concept of Honeyd

Honeyd is a low interaction virtual honeypot. It can simulate arbitrary TCP services and supports multiple IP addresses. It can adopt different operating system personalities. Honeyd can read nmap fingerprint file and can fool nmap. It also supports FIN scan policy. It is fairly portable and can compile and run on BSD, Linux or Solaris. Honeyd is mainly depended on 2 libraries: libevent and libdnet. Honeyd never intercepts the network traffic. All the traffic needs to be directed towards honeyd. With the use of Arpd, honeyd can use IP addresses on the existing network. Arpd is also capable of determining all the unallocated IP addresses (Provos, 2003).

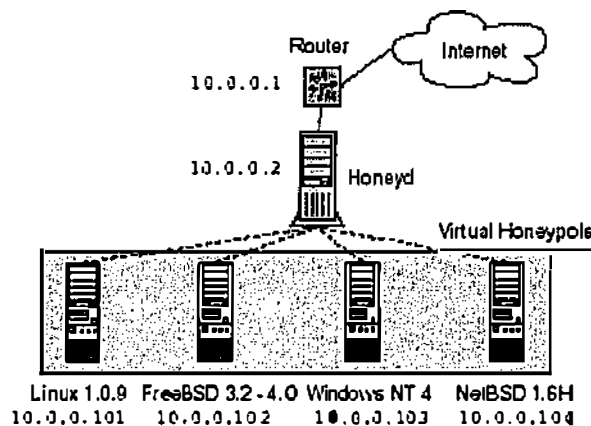


Figure 2.14: Traffic to Honeyd and its virtual honeypots

(Provos, 2003)

The address resolution protocol (ARP) is a protocol used by the Internet Protocol (IP) network layer protocol to map IP network addresses to the hardware addresses used by a data link protocol. The term address resolution refers to the process of finding an address of a computer in a network. Therefore, ARP is provided that will translate the IP address to the physical address of the destination host. It uses a lookup table or ARP cache to perform this translation. When the address is not found in the ARP cache, a broadcast is sent out on the network, with a special format called the *ARP request*. If one of the machines on the network recognizes its own IP address in the request, it will send an *ARP reply* back to the requesting host. The reply will contain the physical hardware address of the host and source route information (if the packet has crossed bridges on its path). Both this address and the source route information are stored in the ARP cache of the requesting host. All subsequent packets to this

destination IP address can now be translated to a physical address, which is used by the device driver to send out the packet on the network. (Comer, 1995, p73-81)

Instead of simulating flat networks, honeyd supports the creation of virtual topologies:

```
route entry 10.0.0.1
route 10.0.0.1 link 10.2.0.0/24
route 10.0.0.1 add net 10.3.0.0/16 10.3.0.1 latency 8ms loss 1.5
route 10.3.0.1 link 10.3.0.0/24
route 10.3.0.1 add net 10.3.1.0/24 10.3.1.1 latency 7ms loss 0.5
route 10.3.1.1 link 10.3.1.0/24
```

With packet delay and loss it is possible to create network topology that seems real.

2.3.4.1 Honeyd Architecture

This section will give an overview of Honeyd's (Provos, 2003) architecture; see figure 4.10. The central dispatcher gets all packets which callbacks for TCP, ICMP or UDP. Other protocols cause packet to be dropped silently. The dispatcher checks the length of the IP packet and verifies its checksum. The dispatcher queries the configuration database for a honeypot configuration that corresponds to the destination IP address. If no such configuration exists, the default template is used. All output packets passes through the personality engine, where various personalities are for various operating systems. Packets are adjusted according to configured personalities. ICMP and UDP handlers are very simple in compare to TCP handler. The ICMP handler replies to Echo Request only and UDP handler knows closed and open status. Closed UDP port generates ICMP Port Unreachable message. The TCP handler requires complete TCP state machine. The State machine tries to be as correct as possible. Initially ports may be in Listen or Closed state. Listen state causes three way handshake and all the other required TCP states whereas closed state causes a TCP Reset to be sent (*ibid*, 2003).

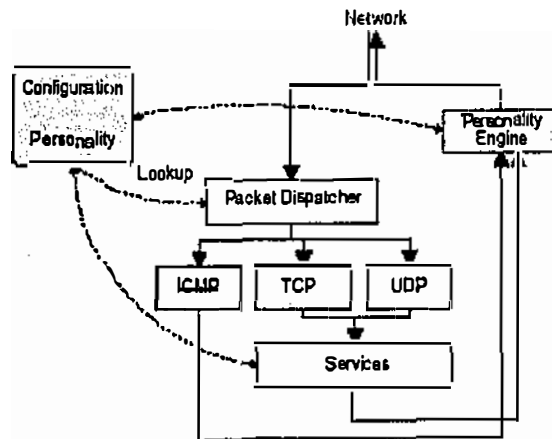


Figure 2.15: Honeyd Architecture

(Provos, 2003)

TCP ports may be connected to an arbitrary service:

```
add template tcp port 80 "sh scripts/web.sh"
```

Honeyd forks child process to execute service applications. Stdin/out is connected via socket pair to TCP stream. Stderr is just a pipe and used for service syslog. Libevent handles incoming and outgoing data. New I/O channels just get added without extra complication for the application. While implementing honeyd, TCP ports may be proxy connected to another machine

```
add template tcp port 23 proxy 10.23.1.2:23
```

This allows the honeyd administrator to simulate services for a given personality via proxy redirection. In this honeyd opens a regular TCP connection, Stdin/out of that socket is connected to honeyd TCP stream and Libevent takes care of the rest.

2.3.4.2 Personalities

Every packet generated by honeyd passes through the personality engine which introduces operating system specific quirks into the packets for nmap identification. Personalities are defined via nmap fingerprint file (*nmap.prints*).

2.3.4.3 Libraries

Honeyd is mainly supported by various libraries (Provos, 2002):

- ❖ Dug Song's *libdnet* – dumb network library (Snog, n.d)

Provides a simplified, portable interface to several low-level networking routines, including network address manipulation, kernel arp(4) cache and route (4) table lookup and manipulation, network firewall implementation, network interface lookup and manipulation and raw IP packet and Ethernet frame transmission.

- ❖ *Libevent* – event notifications on file descriptors (Provos, 2002 (a))

The event API provides a mechanism to execute a function when a specific even on a file descriptor occurs or after at a given time has passed. *libevent* is meant to replace the asynchronous event loop found in event driven network servers. An application just needs to call *event_dispatch()* and can then add or remove events dynamically without having to change the event loop.

2.3.5 Methods for deployment of Honeypots

There are four different methods of honeypot deployment. These include Deception Services, Weakened Systems, Hardened Systems and User Mode Servers. Each of these methods are described below (Brenton, n.d):

2.3.5.1 Deception Services

Deception Services are specially designed to listen on a IP service port and respond to inbound network requests. They can be used to emulate common TCP/UDP services for example, Sendmail an SMTP mailer used primarily on the UNIX platfonn. When an attacker connects to the honeypot, he or she may receive a banner, a small piece of infonnation displayed regarding the information of the application, which identifies the service as being some version of Sendmail. This piece of infonnation could be about the developer and version of the particular application. If the attacker is fooled by the deception, he or she may try to gain access to the system. This may allow the administrator to log the efforts of the attacker and safeguard the real production systems who may be running Sendmail (Brenton, n.d).

Deception Services too have some problems. Firstly, it is often difficult to emulate a service that can consistently fool the attacker. For example, the attacker may try various control commands to check for expected responses an attacker will often use lesser known or more esoteric commands to probe systems based on intelligence gathered by initial scans. Unless the deception service is capable of handling such testing, the attacker may become aware of deception and never carry out any attack on the system.

Secondly, deception services can only collect a limited amount of information. It records the initial attempt of attack, but nothing more than that. It may have been useful if it could record the successful attack information, such as other systems compromised using honeypot, or attacker's identity or tools. Since the deception service should not provide any additional access to the machine, additional forensic information cannot be collected (*ibid, n.d*).

Finally, on a theoretical basis, deception services may have some form of vulnerability that could give attacker unexpected access to the system. For example, if the honeypot is compromised, the attacker may be able to remove all the evidence of their attack such as log files. If the honeypot is compromised it may be used as a launching pad to attack other machines or networks.

2.3.5.2 Weakened Systems

The weakened system deployment involves installation of an operating system with some known vulnerabilities. This makes it easier for the attacker to penetrate into the system and thus helps in collecting data about the attacks carried by the attacker. Logs are normally saved on a remote system, in case the attacker erases the hard disk before leaving the system. There are also memory dumps stored on the system. These memory dumps stores information about the system and applications used. These can be retrieved and stored remotely by taking a memory dump of the system using some automated executable scripts (*ibid, n.d*).

One of the benefit of this approach is that the actual services may be used which an attacker may try to exploit. Weakened systems provide a lot of information about the attack as the actual vulnerability is exploited by the attacker. This may provide information about the attacker or their tools used to exploit the services. Such information may be used to check the main production servers or systems to determine the vulnerabilities exploited by the attacker on a weakened system.

Weakened systems require very high maintenance and with very little return (Brenton, n.d). If the vulnerability has been on the operating system for quite a long time and the administrator knows about it and its patches, then it is not of much help in data collection. Also an attacker may become suspicious seeing a very obvious vulnerability not patched on the network. Hence, weakened systems can be of little value for data collection and analysis as it is depended on known security exploits and their patches (*ibid, n.d*).

2.3.5.3 Hardened Systems

In a hardened system deployment, the base operating system is patched for all of the known vulnerabilities. These vulnerabilities are published on various security related web portals and also alerts are released by various vendors about the vulnerabilities associated with their products as they are identified. The system is made secure from the known vulnerabilities and then left for the attackers to attack. If an attacker is able to compromise such hardened systems then forensic evidence collected from the attack may be useful for further improving the systems security and also can be used for law enforcement purposes. Such method reduces the maintenance time and increases the data collection from the system. It also is a good method for providing new attack intelligence on unknown attacks as the system is hardened against known vulnerabilities (Brenton, n.d).

Such hardened systems do have a weakness. These systems rely on the system administrator skills. If the attacker possesses greater and more efficient skills than the system administrator, then it will be easier for him to compromise the system and may also remove their traces from the systems which could lead to them. The compromised honeypot may also be used as a launching pad for carrying attacks on

various other systems within the same network as well other networks over the Internet.

2.3.5.4 User Mode Servers

User Mode servers are fully functional servers that have been nested within the application space of the host operating system. When an Internet user transmits a request to the IP address of one of the user mode servers such as Windows NT, the host accepts the request and routes it to the proper user mode instance such as word processor, excel spreadsheet, email program, etc... For a user on the Internet, user mode host would appear like a router or firewall. This could fool the attacker as it would pretend to be a main network of the organisation. User mode servers are not available for every operating system. To create a user mode server you must start with a regular operating system and port it so that it can be run as a user application. They are mainly available for operating systems like Linux and NT, so it limits the choice of operating system when deploying the honeypot. But Windows NT and Linux are the most widely used operating systems so this may be an advantage of using User mode servers for deploying honeypots.

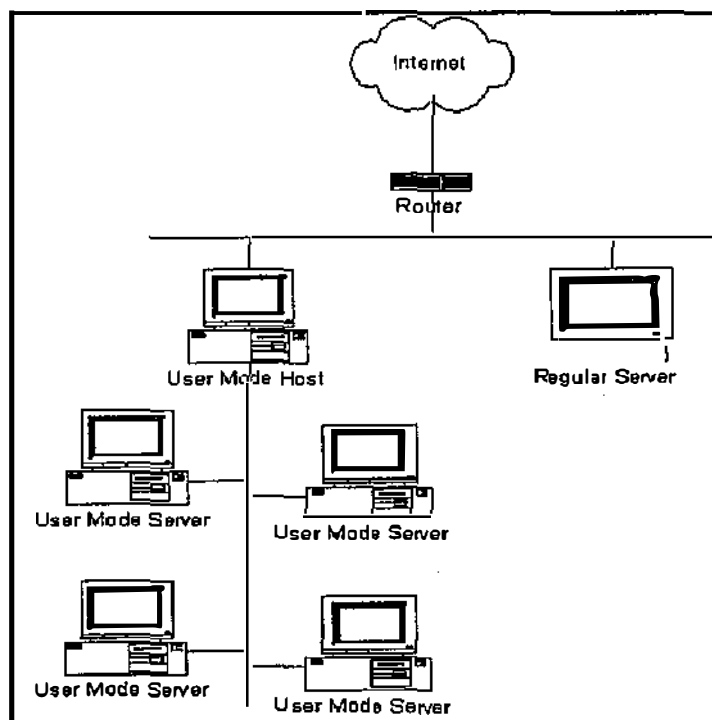


Figure 2.16: User Mode Servers

(Brenton, n.d)

Figure 2.16 shows a user mode server in a network environment. For any user over the internet, user mode host will appear as a router or firewall while other user mode servers will appear to be individual hosts connected on a subnet. This seems to be a legitimate network structure as it is common that individual hosts are protected by the firewall or router in a subnet. It is also possible to use proxy ARP to connect user mode servers with the same logical network segment. This may provide opportunity to hide the honeypots on a network (*ibid*, n.d).

2.3.6 How Honeynet is created?

A highly controlled network using appropriate security perimeters such as firewalls, routers, intrusion detection systems, etc... is created. Within this network honeypots are placed and then the activities of attackers are monitored, captured and analysed. Any traffic to honeypots is suspicious by nature. Such traffic to honeypots could be stealthy port scan of a single service or brute force probe of the entire network.

“Creating and maintaining a successful Honeynet depends on two critical elements: data control and data capture.” [Anonymous, 2002a, p20]

2.3.6.1 Data Control

Data Control is the inbound and outbound control of data flowing through a network to the system. Once the honeypot within the Honeynet is compromised, the administrators have a responsibility to ensure that the honeypots are not used to compromise other systems within the network or other outside networks. A firewall can act as a suitable access control device for data control on a honeypot. A firewall can be used to separate the Honeynet from rest of the network or Internet (see Figure 2.12). All inbound and outbound data should flow through the firewall. All inbound and outbound traffic must be controlled in an automated fashion, without the attacker getting suspicious. Therefore, a transparent firewall can be used.

It is very important that all traffic should go through firewall. The firewall controls the flow of traffic by defining three main rules (Anonymous, 2002a):

1. Anyone can initiate a connection from the Internet to the Honeynet. This will allow the attackers to scan, probe and exploit systems on the Honeynet.
2. The firewall controls how the honeypots can initiate connections to the Internet. If no outbound traffic is allowed from the compromised Honeypot to the Internet, the attacker will become suspicious and may leave the network. Therefore, some limited outbound connections should be allowed. The Honeynet Project found that five to ten outbound connections within a 24-hour period should be sufficient (Anonymous, 2002b). This will give enough connections to the attackers to download their tools, connect to IRC or whatever activity they require to do. This will give more opportunity to learn from attacker's activities but goal should be to contain the traffic initiated from the Honeynet to the Internet or other networks as Honeynet can be used as a launching pad for attacks on other networks.
3. The Honeynet and the internal network should not have any direct communication. The internal network is used for critical data collection for all data generated by the attacker's activity. If the Honeynet is compromised, it will prevent to communicate with the internal network and modify or delete any data collected.

There are other access control systems which can be used for the secure means of data flow over the network. For example, use of Virtual Private Networks (VPN), Routing table configuration over the router for data flow can be used to ensure secure data flow. Therefore, to control and contain data flow to and from the Honeynet, access control must be wisely used to separate Honeynet from other networks.

2.3.6.2 Data Capture

Data capture is the collecting of all activity that occurs within the Honeynet, including both at the network, for example with the use of intrusion detection systems, and system levels using log files, keystroke loggers, etc.... Data capture is critical for the success of a project. According to the Honeynet project (Anonymous, 2002b), it is ideal and useful to capture data in multiple layers within a honeynet design. It is a challenge to capture the maximum data possible without the attacker getting alerted of his or her actions. To have an efficient data capture mechanism multiple data capture

methods should be deployed within the network. If a single layer of data capture fails, there is always a backup layer available to alert the administrator about the fact that a system was compromised. As discussed in previous section 2.3.6.1, how firewall can be used for data control, firewall can also be used to capture data. Firewall can be an excellent data capture layer as all the network traffic must flow through it. Although information captured by firewall is very limited as it cannot capture any keystrokes or packet payloads but it is capable of logging the attack timestamps, source and destination IP addresses and also source and destination port numbers. Another good example for capturing data could be the use of IDS systems. It is always useful to use multiple layers for capturing data. In case the honeypot is compromised by the attacker and he or she gain access to one of the data capturing mechanism then there are always other sources to look back for data analysis purpose (Spitzner, 2003).

It is essential to note that none of the captured data is stored on the honeypot locally. Any information stored locally can potentially be detected by the attacker. All data captured on the honeypot should be transferred to a secure location on the network where no other communication is allowed with the Honeynet. Methods such as use of virtual private networks (VPNs) or IDS systems can be used for remote logging of the data captured. If the honeypot is detected, data stored locally on the Honeynet can be modified or destroyed by attackers. This may result in the failure of deploying Honeynet in a network and may not fulfil the actual purpose of deploying them. Therefore, securing the stored data is very important in a successful Honeynet.

2.3.7 Conclusion

The chapter outlined the TCP/IP protocol and security weaknesses thereof. It is important that these principles and protocols are understood so that effective honeypot design can be undertaken.

Honeypots are unique security tool which are intended to be probed, attacked or compromised. Their purpose is to protect the main production network against the attackers. This chapter provided with solid definition of honeypot, types of honeypots, their advantages and disadvantages and methods of deployment of honeypots.

In the next chapter, the researcher describes the methodology used for the research.

3. Methodology

3.1 Introduction

This chapter discusses the overall research framework adopted for this research. Section 3.2 presents a brief literature review on various research designs. Section 3.3 discusses situations in which laboratory experiments can be an appropriate method of research. Section 3.4 explains the complete research process adopted by the researcher detailing each stage of the research. Section 3.5 illustrates various elements of the laboratory study for the purpose of this research. Section 3.6 justifies why this particular research method was adopted. Section 3.7 evaluates the overall research methodologies by presenting the advantages and disadvantages. Finally, section 3.8 concludes the chapter.

3.2 Literature review on research design

There are various issues which need to be discussed before designing a research methodology:

- What is the Philosophical perspective of the underlying research (e.g., Positivist, Interpretive and Critical)?
- What methodology or strategy should be adopted to link various research methods (e.g., experimental research, survey research, ethnography, etc.)?
- What methods – techniques and procedures - should be used (e.g., questionnaire, interviews, observations, etc.)?

Philosophical perspective of the research

Research is always based on some underlying assumptions about the validity of research and appropriateness of the research method used. Therefore it is very important to know these assumptions while conducting or evaluating research. The

most relevant philosophical assumptions are those which relate to knowledge and how it is being obtained, termed as *epistemology* (Myers, 1997).

Orlikowski and Baroudi (1991), following Chua (1986), suggested three categories, based on the underlying research epistemology: Positivist, Interpretive and Critical.

Positivist takes the form of theoretical propositions, which can be stated mathematically or verbally. Positivist research generally attempts to test theory in an attempt to increase the predictive understanding of a phenomenon. Orlikowski and Baroudi (1991, p5) classified information science research as positivist if there was evidence of formal propositions, quantifiable measures of variables, hypothesis testing and drawing of inferences about a phenomenon. Positivist also involves empirical observation and measurement of data for theory verification (Creswell, 2003). Interpretive research helps in understanding human thought and action in social and organisational context. It attempts to understand phenomena through the meanings that people assign to them and interpretive methods are aimed at producing an understanding of the context of the information science. Interpretive study does not predefine dependent and independent variables but focuses as the research progresses (Galliers, 1987). It does not have to be limited by hypothesis testing and tight experimental control. Instead, the external validity of the actual research question and its relevance to practice is emphasized, rather than restricting the focus to what is researchable by rigorous methods. Critical research focuses on the oppositions, conflicts and contradictions in contemporary society (Myers, 1997). It denotes a critical process of inquiry that seeks to achieve emancipatory social change by going beyond the apparent to reveal hidden agendas, concealed inequalities and tacit manipulation involved in a complex relationship between IS and their social, political and organisational contexts. It aims to reveal interests and agendas of privileged groups and the way they are supported or protected by a particular information system design or use.

Types of Research Methodologies

Research methodology can be classified in various ways. However, it is common to classify them into qualitative, quantitative and mixed methods.

Quantitative	Qualitative	Mixed Methods
<ul style="list-style-type: none">• Experimental design• Non-experimental designs e.g., surveys	<ul style="list-style-type: none">• Narrative• Phenomenology• Ethnography• Grounded theory• Case studies	<ul style="list-style-type: none">• Sequential• Concurrent• Transformative

Table 3.1: Research Methodologies

(Creswell, 2003)

Types of Research Methods of data collection and analysis

Quantitative Research Methods	Qualitative Research Methods	Mixed Methods Research Methods
<ul style="list-style-type: none">• Predetermined• Instrument based questions• Performance data, attitude data, observational data and census data• Statistical analysis	<ul style="list-style-type: none">• Emerging methods• Open-ended questions• Interview data, observation data, document data and audio visual data• Text and image analysis	<ul style="list-style-type: none">• Both predetermined and emerging methods• Both open- and closed-ended questions• Multiple forms of data drawing on all possibilities• Statistical and text analysis

Table 3.2: Research Methods

(Creswell, 2003)

Creswell (2003) suggests that the philosophical view, methodologies or strategies and methods all contribute together to the research approach. On the basis of the above information, the three research methodologies may be defined as (Gupta, 2003):

- A quantitative approach is one in which the investigator primarily uses positivist claims for developing research, employs strategies of inquiry such as experiments and surveys and collects data on predetermined instruments that yield statistical data.
- A qualitative approach is one in which the research often makes research claims based primarily on constructivist perspectives employs strategies of

inquiry such as ethnographies, grounded theory studies, or case studies and collects open-ended, emerging data with the primary intent of developing themes from the data.

- A mixed approach is one in which the researcher tends to base research claims on pragmatic grounds, employs strategies of research that involve collecting data either simultaneously or sequentially to best understand research problems and data collection represents both quantitative and qualitative information.

For the purpose of this research, the researcher has adopted a mixed-method approach which was empirical in nature. In a mixed method approach each technique produces different empirical data which can complement data collected from another technique. A large amount of quantitative data (such as total number of alerts, various source and destination IP addresses, frequency of port numbers probed, etc...) was collected using the laboratory experimental method which was analysed and interpreted using appropriate qualitative and quantitative methods. Concepts and understanding of the problem situation was developed from the observations and patterns noticed in the data collected. Therefore the research is best described as empirical learning approach using laboratory experiments for observing data patterns.

3.3 When are laboratory experiments used?

Universalistic versus Particularistic research goals

Universalistic research (Kruglanski, 1975, p. 105) is used to investigate theoretically predicted relationships between abstractly specified variables, for example: pornography and sex crimes. It is important to conduct tests on such hypotheses as they provide light on the validity of the theory from which the hypothesis was derived. On the other hand, particularistic research involves various quantitative questions such as, how much, how often or how many. Such particular questions presuppose a population, setting and other conditions in regard to the applicability of the research findings.

When the research goals are universalistic, since a particular setting and population of participants are not crucial aspects of the hypothesis under study, a laboratory study may be appropriate. When the research goals are particularistic a laboratory study is less suitable as it will be difficult to answer questions about the impact of a program in a specific setting with specific people unless the research is conducted under those same conditions.

For the purpose of this research, it can be classified under particularistic research as similar network architecture was implemented at each iteration of the research and was tested by various attackers. These attackers were anonymously chosen as described in later section.

Examining what does happen versus what can happen

Sometimes the goal of the research is to determine what *can* happen under such situations or circumstances (Judd, 1991). For example, in theoretical based hypothesis a set of circumstances are specified and the goal is to create those circumstances in order to determine if the predicted phenomenon can be observed. Results for such hypothesis are already predicted but it is the validity of such predictions that needs to be proved by constructing such circumstances, which may be possible to do in a laboratory environment.

With regard to this research, the theory of deception was already applied by various deceptive techniques and tools. What needed to be determined was the effectiveness of the actual deception by using such tools and techniques in a virtual networked environment. It was not possible to do this network testing in a real world by connecting it to the external networks for several reasons. Firstly, it was essential to determine if the designed network is capable of deceiving attackers. Secondly, if the designed network is also secure enough to handle the attacks otherwise it could have been used as a launching pad for carrying out attacks on other networks if it was done in an open network or laboratory situation. Therefore, laboratory research method seemed to be more appropriate as it provided a controlled environment to protect the external networks and also provided the author with valuable and controlled results.

3.4 Research Process

The research process has been divided in various stages where results of each stage depend on the previous stage. The figure 3.1 below explains each stage of the research process.

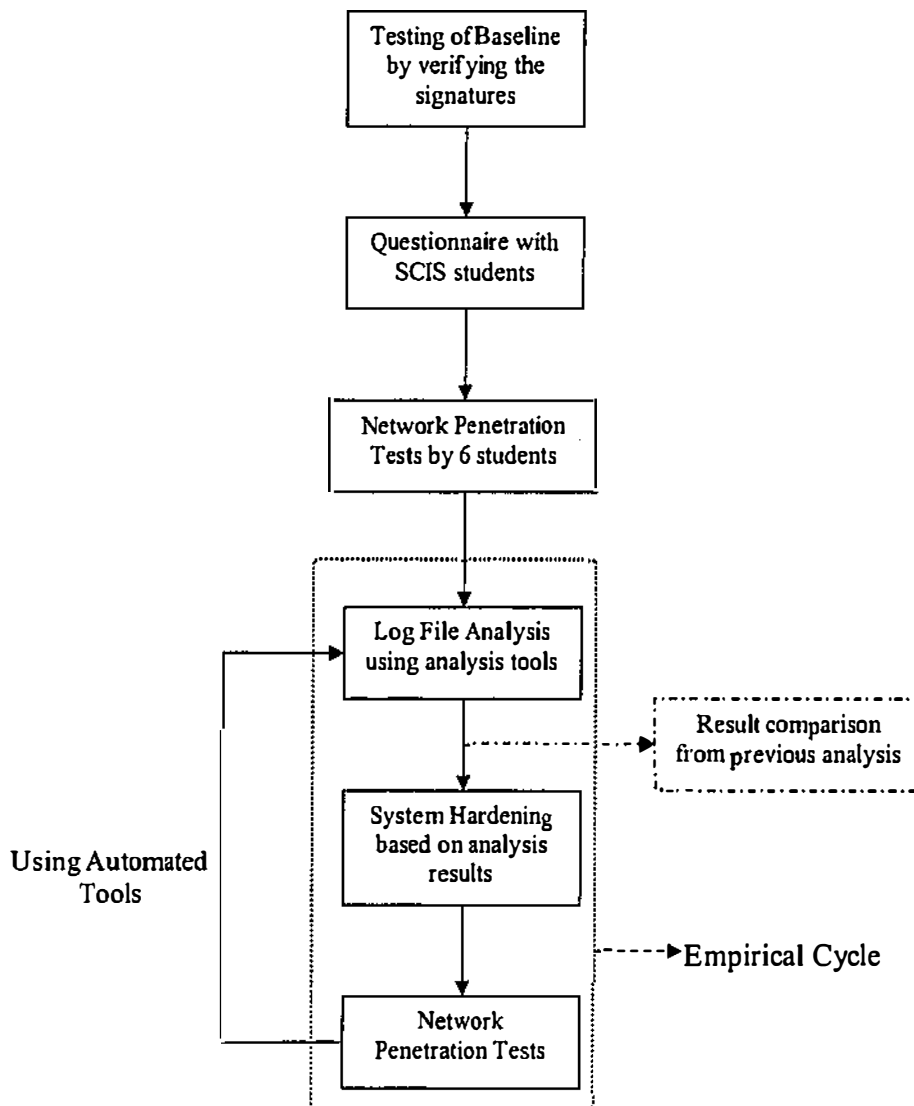


Figure 3.1: Theoretical Framework: Stages of Research Process

Stage 1: Verification of Signatures

In the initial stage of the research, a baseline system was created using the Honeyd, an open source honeypot. Honeyd is provided with the list of TCP/IP signatures of various operating systems and networking devices (around more than 400) which can be used for configuring the honeynet. But it was not known that the signatures

provided with the honeyd can be successfully used. Therefore, before implementing the honeypot for the data collection it was necessary to verify these signatures using honeyd. Each individual signature was implemented in the honeyd configuration file and Nmap (a network scanning tool) was used to test them. All the successful signatures (nearly 50% of the total signatures) were listed separately and were then used for baseline the honeypot. Once the baseline signatures were achieved, the researcher moved to the next stage of the research.

Stage2: Questionnaire

Selected attackers were required to attack the system on which Honeyd was installed, in order to test its effectiveness. A *Questionnaire*, a type of survey method, was chosen for selecting the attackers. A preset questionnaire was prepared with 20 questions related to computer security. The primary aim of this questionnaire was to select students for network penetration testing. It was conducted among the students of School of Computer and Information Science (SCIS) of Edith Cowan University, Perth. These students were chosen primarily because they were easy sample group to approach within the computer science school and were computer literate. So there was high possibility among these students having some knowledge of operating systems and computer systems which was essential for participating into such exercise. Students who attempted the questionnaire were not made aware of the real nature of the project. They were informed only about the hacking exercise. They were told that the exercise was only to determine their hacking skills. It was purely to preserve the nature of the project whose main aim was to disguise attackers. The effectiveness of hackers was unknown until they performed the network penetration tests. They were purely selected on the basis of their performance in computer security literacy test. Due to the lack of resources and time it was not possible for conducting any such practical exercise which could help in determining the effectiveness of the hackers. This would have been an advantage to the research results but unfortunately was not possible. The prototype of the computer security literacy test was also approved and validated by two network consultants. The two network consultants were Craig Valli, Edith Cowan University, Australia and Andrew Dawson, British Petroleum, London. None of the students, who attempted the questionnaire, were identified by their names or anything else that could identify them. They were assigned a subject number in order to preserve their anonymity.

The designed questionnaire was attempted by 10 students, from which top 6 were selected for the next stage of the project i.e. for network penetration exercise. These students were selected primarily because of their excellent performance in the questionnaire in which they scored 100%. The total number of students selected was restricted because of limited resources in terms of availability of computer systems, availability of appropriate laboratory, and also time constraints.

Stage 3: Network Penetration Tests by selected students

The selected students conducted the network penetration tests for the configured Honeynet. These tests were conducted to probe the network for exploitable vulnerabilities. As mentioned earlier, these students were not aware of the actual Honeynet in the network structure. These students were allowed to bring in their own laptops and programs required by them for network testing. This provided an opportunity for the students to use their own desirable tools to do the exercise. Also it was beneficial for the purpose of research to learn about various attacking tools which can be used against such network environment. All the results obtained during the testing were kept completely anonymous. The Network Penetration exercise was conducted continuous for several days to provide enough time to participants to probe the network.

There was only one single laboratory available for the purpose of the network penetration exercise, therefore all the selected students were asked to take their places within that laboratory. All of them had their own laptops with them so it was not required to provide them with separate machines. Only a couple of them requested for access to an extra machine, which was provided to them.

Once the network penetration exercise was completed by the participants, they voluntarily provided their anonymous reports to the researcher. These reports outlined their point of views regarding the network architecture and their findings during the exercise.

Stage 4: Log files Analysis

Logs generated during the above network penetration exercises were stored securely on the server and also appropriate backup measures were adopted, such as storage on zip disks, database backup on remote system. Various analysis tools were used to carefully analyse these log files. The tools used were:

- **ACID (Analysis Console of Intrusion Databases)**

ACID was primarily used in conducting statistical analysis of the log files. This tool helped in classifying the security alerts generated during the network penetration tests conducted by the participants. ACID allowed the researcher to analyse the data using options such as most frequent alerts, top 10 source IP addresses, top 10 destination IP addresses, alerts related to individual port numbers, etc.

- **Ethereal and tcpdump**

Log analysis also involved study of all malicious packets that came through the system. Their structure and their potential threats were carefully analysed. The malicious packets were analysed using Ethereal Packet analysing tool and tcpdump. Ethereal was mainly used as it has the ability to (Anonymous, n.d(f)):

- Examine data from a live network or from a capture file on disk
- View the reconstructed stream of TCP session
- Can be used for both Windows and Unix machines
- Dissect 280 protocols
- Selectively highlight and colour packet summary information
- Save all our parts of captured network trace to disk

Ethereal allowed the researcher to generate the hierarchical statistic of the protocols. Using ethereal researcher was able to analyse each individual captured packet separately.

- **Analyst Notebook 6**

Analyst Notebook 6 was a commercial analysing tool which helped in generating visual diagrams of the network traffic captured in the log files. It helped in

interpreting complex information through visual diagrams, which was not possible doing manually. The results generated by Analyst were easy to understand and accessible.

Also some of the log files were analysed manually. During the analysis, frequencies of various types of attacks and their effects over the network were identified. These tools were selected because they were easily available as freeware (except Analyst) over the internet and were also open source programs. This provided enough flexibility in their configuration according to the research requirement. From the collective combination of results obtained from these analysing tools, various network configuration weaknesses were identified. These weaknesses were identified by observing the data patterns within the log files and also by understanding the tools or programs used by the participants for attacking the network. The information about these tools or programs was captured within the logs. From the detailed analysis of the logs and captured packets, it was identified that there were few configuration problems in mimicking the Honeynet. It was also found that some services, which were prone for attack, were also not configured properly. Such weaknesses were identified with the help of using various analysing tools described above.

Stage 5: System Hardening

Based on the results obtained in the above analysis stage, system and network configuration was improved and hardened for further testing and data collection. System hardening was done by improving the configuration file of the honeypot using more specific signatures and also by improving the configuration files of the services emulated on various ports.

Stage 6: Network Penetration tests using Tools

Once the hardening of network and system was completed, further network penetration testing was conducted using some automated security tools available freely over the internet. These tools were mainly used because they were freeware. These testings were conducted by the researcher using various hacking tools. This testing was done to determine if the behaviour of the network has improved after the system hardening based on the weaknesses identified in previous analysis. After the completion of testing, steps in stage 4, 5 and 6 further followed in order to determine

if the level of deception in the Honeynet has improved or not. Results obtained from each analysis stage were compared with the previous analysis results. This comparison of results helped in determining the level of deception. This was determined by studying the various successful hacking techniques, identified in the log files, which were lower in number and less effective then in comparison to the first analysis conducted after the initial penetration tests. Steps in stage 4, 5 and 6 were empirically continued until it provided minimal opportunity to the attackers to determine security holes in the network. With the continuous improvement in the Honeynet, it may be considered that a successful level of deception was achieved which was effectively able to deceive the attackers.

3.5 Elements of the laboratory experiments

This research is based on the findings during the experiments conducted in a closed laboratory environment. The uniqueness of laboratory research lies in its flexibility. It gives the researcher the ability to construct physical settings, tasks, social environments, and many other factors that influence human behaviour. But also laboratory research has drawbacks. One of the drawbacks of laboratory research is its artificial nature. Laboratory research is artificial in nature as it permits the creation of settings or tasks and manipulations which assists in achieving the required research goals. The reliability of the results may depend on various factors, such as sample size, depending on the type of any research (Judd, 1991; Agassi, 1992; Fujita, 1996). In this research, the sample size used, in terms of number of participants, was small.

The sample size constraint was difficult to overcome for several reasons. Firstly, it was difficult to find people with an appropriate amount of computer security knowledge to participate in this type of research. But this does not imply that the results obtained in a laboratory research are wrong or incorrect. Those results are valid under the created situation within the laboratory and from which various assumptions may be made but it may not be applied to large population due to the small sample size reducing reliability of results.

Participants' awareness of the research was very important for the purpose of this research. In a laboratory research method, it is sometimes difficult to keep people unaware of the fact that they are participating in research. This may sometimes create problems for the validity of the research (Judd, 1991). The participants were selected for conducting the penetration tests in the laboratory were unaware of the true nature of the research. They were selected through a computer literacy test to conduct the hacking exercise in a closed laboratory environment. They were not informed about the deception implemented in the network and this was the main focus of the research. These participants, while they were conducting penetration tests, were remotely monitored by the researcher. Their moves, tactics and also their personal expressions while doing the exercise were monitored by the researcher in the laboratory.

In a laboratory research methodology, one of the main goals of the research is *Control*. Control or the minimization of irrelevant influences on whatever the research is designed to investigate, is made possible by the laboratory's isolation from external influences (Judd, 1991). For the purpose of this research, a closed laboratory environment was adopted, isolated not only physically but also electronically. This laboratory research was carried out in an isolated network environment with no communication with any other network. This control environment was beneficial for the purpose of the research because it prevented from distractions and influences caused by external sources. By removing external variables and their impacts by using the closed laboratory the possibility of participants having their work affected in direct or indirect manner was greatly reduced. Also having a control isolated network environment gave a sense of security while carrying out this research to the attackers.

3.6 Rationale of the Research

This section of the chapter provides the justification of the research process adopted. The researcher justifies why he chose Mixed-method positivist approach towards undertaking a survey method (e.g. Questionnaire) and observational study respectively.

Philosophical View – Positivist

This research depends on the tests and statistical conclusions drawn from the tests conducted within the laboratory. There was no previous data from any other research which was being interpreted or critically analysed or used in anyway.

Research Methodology – Mixed Method

Both quantitative (e.g. experiments, questionnaire) and qualitative (e.g. observational study) methods were adopted. Mixed Method approach was mainly used:

- To achieve a higher degree of validity and reliability
- To overcome the deficiencies of single method studies
- To help in explaining the findings of another method.

Research Methods

• Laboratory Experiments

This research was being particularistic in nature as a particular network was probed and attacked by selected attackers. Also it provided a control and flexible environment for the purpose of the research. According to the researcher, this was the only possible method to adopt which could have justified the nature of the underlying research.

• Questionnaire

The Questionnaire was primarily chosen because of following reason:

- Offers greater assurance of anonymity
- Could be completed at the respondent's convenience
- Questionnaire are stable, consistent and uniform measure, without variation
- Helps in avoiding bias selection of participants

• Observational Study

According to the researcher, this research was an observatory study. Participants were not made aware of the deceptive network created by the researcher. They were just asked to test their hacking skills over a local network. This exercise may have just a minimal impact over the participants. From the data collected, the

researcher tried to observe various data patterns and tools and tactics used by participants.

3.7 Evaluation of research methodology

However the researcher has justified his adopted methodology in the previous section, but all these methodologies and methods do have some advantages and disadvantages. These are briefly summarised below:

Advantages and Disadvantages of Questionnaires

Advantages

Questionnaire is a type of survey method which provides a type and amount of information that other methods cannot provide. There are various reasons that make them one of the popular methods used in social science research. Questionnaires are less expensive than other methods and also produce quick results. They are also very convenient method as they can be completed at the respondent's convenience and offers greater assurance of anonymity. They promise a wider coverage of people, since they can approach respondents more easily than other methods. They also help in avoiding bias or errors caused by the presence or attitude of the researcher. (Sarantakos, 1993)

Disadvantages

There are various limitations associated with the nature of the questionnaires. They do not allow probing, prompting and clarification of questions. It is not possible for the researcher to know whether the right person has answered the questionnaire also if the respondents have followed the right order of the questions. Due to lack of supervision, partial response is quite possible. (Sarantakos, 1993)

Advantages and Disadvantages of Laboratory Experiments

Advantages

The laboratory is a unique setting for research in many ways. It provides a *Control* environment which minimizes various external influences which could affect the

underlying investigation. *Manipulation* of independent variables is also facilitated by the laboratory research settings. Laboratory experiments also allow the researcher to setup various settings which may influence the research participants. The flexibility of the laboratory research allows the researcher to attain control, implement a manipulation and construct an appropriate setting in different ways depending on the nature of the underlying research (Judd, 1991).

Disadvantages

Laboratory experiments also have various limitations which may affect the research. In a laboratory settings there are chances of human errors. It is an artificial environment created by the researcher so it is very prone of errors made by the researcher. Also the sample used in the experiments too has a limitation. It may not be a representative of the population (Judd, 1991; Anonymous, n.d.(d)).

Advantages and Disadvantages of Observational Study

Advantages

Observation provides information when other methods are not effective. It offers information without relying on reports of others. Observation allows the collection of a wide range of information even when this information is thought to be irrelevant at the time of study. It is relatively inexpensive mode of study. (Sarantakos, 1993)

Disadvantages

Observation study too has various limitations. They cannot be applied when large group are studied. It is also exposed to the observer's bias, selective perception and selective memory. Observation cannot offer quantitative generalisations on the results. It is relatively laborious and time consuming method. (Sarantakos, 1993)

3.8 Conclusion

On the basis of the literature reviewed on research design, the researcher had adopted mixed-method positivist approach for the research. During the course of the research

laboratory experiments, questionnaire and observational research method were conducted.

The nature of the research was evolutionary since the findings of one stage were the basis of the focus of next stage of the research. This provided the higher level of granularity among the results. Thus a mixed-method approach was more appropriate as it provided flexibility to him in comparing the results of each stage, generated in the experiment, and then designing the focus of next stage accordingly.

4. Tools for Data Collection and Analysis

There were various tools used for the data collection and analysis purpose. There were tools which were used for collecting raw data and then with the help of some other tools generated some meta data for analysis purpose. This can be summarised in the Figure 4.1 below:

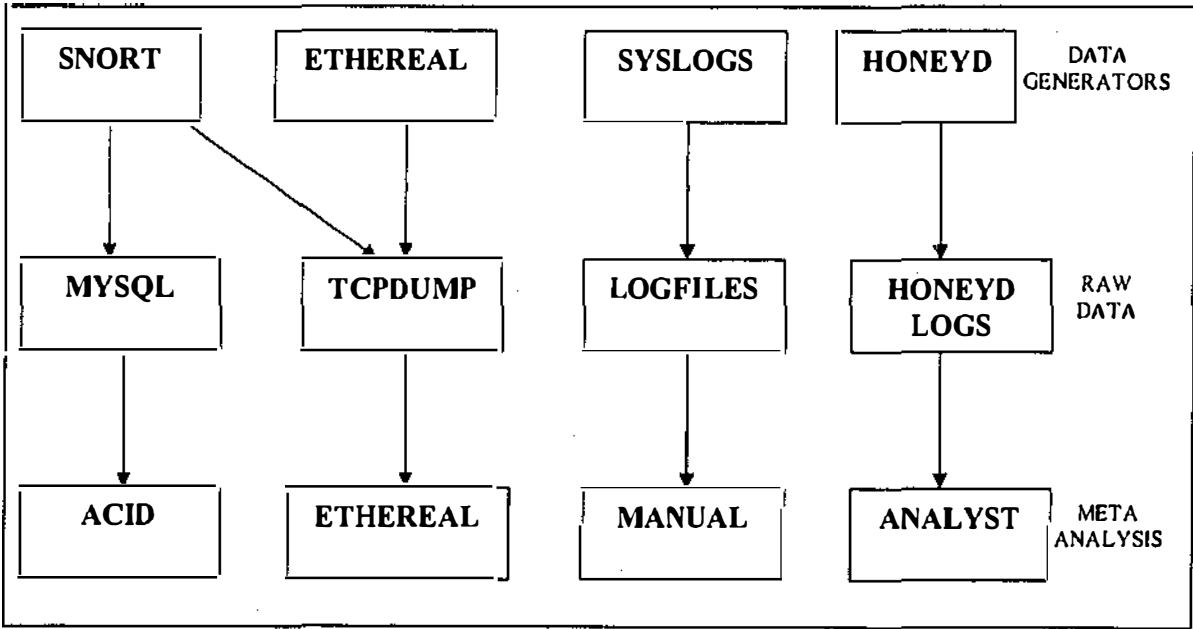


Figure 4.1: Tools for data collection and analysis

4.1 Snort

Snort is a freely available intrusion detection system which can be distributed and modified under the GNU General Public Licence (See Appendix B). It is freely available from <http://www.snort.org>. According to the snort website, it is capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. There are three main modes in which Snort can be configured (Baumann, 2002; Roesch, n.d):

- **Sniffer Mode:** Snort is used as a packet sniffer and can be configured to show only IP headers or the IP payload as well.
- **Logger Mode:** All packets can be logged to a file and inspected later using various tools.
- **Network Intrusion Detection Mode:** All packets are compared to the rule type defined in the configuration files. If any rule matches, the packet is logged and an alert is sent. This is being illustrated in the figure 4.2 below:

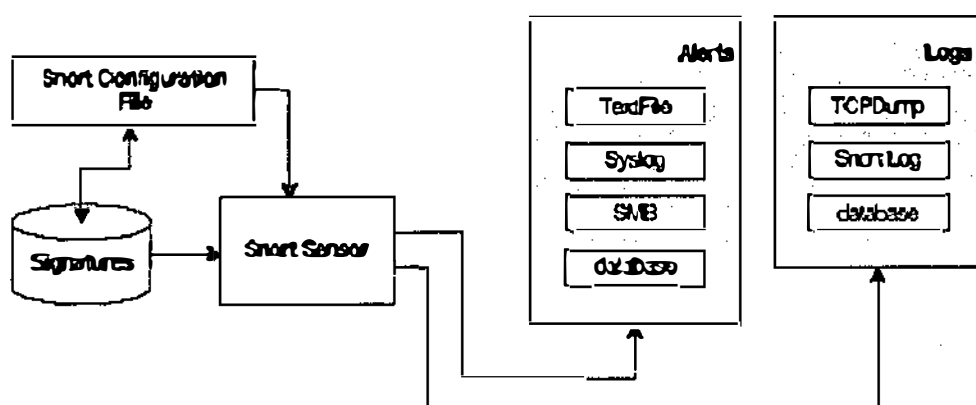


Figure 4.2: Snort Overview

(Baumann, 2002)

Any incoming packet is checked for the rules defined in the snort configuration file which further checks with the signature database to determine the type of packet. If the packet matches with the signature, request is sent to the snort sensor which logs the details of the packet and generates an alert for the purpose of the administrator.

It has a real time alerting mechanism and can also log all the alerts to syslog or various types of logging mechanism such as SQL databases. It features allow logging discrete types of messages on separate log files.

Snort is running as a UNIX daemon and is configured through a common Unix .conf file (see appendix C.5). It is connected to the MySQL database where all the logs and alerts are stored (see appendix C.7 for MySQL database format). Snort is a core IDS

engine. It does not contain any analysis tools or remote administration GUIs. There are few available front-ends and analysis tools. One of the best known is ACID (Analysis Console for Intrusion Databases) (Danyliw, n.d.). It is a web based analysis tool which is used to inspect Snort data stored into the database.

Snort is a notable IDS with an extensive list of available signatures. It is easy to install and also the database support is available. Snort has a fast and robust engine. When compared to other commercial IDS, Snort is a better choice as it is available for free and signature file can be updated quickly in compare to commercial IDS as those are dependent on their vendor's supply (Roesch, n.d.).

4.2 MySQL

MySQL, the most popular Open Source SQL database, is developed and provided by MySQL AB (Anonymous, n.d(b)). It can be downloaded by anyone from the internet (<http://www.mysql.com>) for free. It is freely distributed and modified under GNU General Public License (see appendix B). MySQL was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. The connectivity, speed, and security make MySQL highly suited for accessing databases on the Internet (Anonymous, n.d(b)).

MySQL provides excellent connectivity with Snort IDS and ACID (see section 4.4 for ACID) for analysis.

Due to its high degree of connectivity with Snort IDS and ACID (Section 4.4) and also support for SSL connection between MySQL client and server which provides an encrypted connection while transferring the data, it was an ideal choice to use this database for this HoneyNet project.

4.3 Webmin

Webmin is a web-based interface for system administration for UNIX. Using any browser that supports tables and forms (and Java for the File Manager module), you can setup user accounts, Apache, DNS, file sharing as well as other services.

Webmin consists of a simple web server, and a number of CGI programs which directly update system files like `/etc/inetd.conf` and `/etc/passwd`. The web server and all CGI programs are written in Perl version 5, and use no non-standard Perl modules.

Webmin is available under the BSD licence. This means that on Linux and other platforms, Webmin may be freely distributed (<http://www.webmin.com>) and modified for commercial and non-commercial use.

Webmin provides a one-to-one graphical interface to nearly every service and action needed to maintain a UNIX system. It is universally accessible, because it only requires a web browser. It can potentially be accessed from anywhere in the world via a network connection. It is simple, concise, and consistent in its presentation across a wide array of differing services, functions, and operating systems. It is predictable, in that it does not modify files unnecessarily or in incompatible ways. Webmin is an excellent tool for both novice and experienced system administrators. As a tool for novices, it can provide a means of getting involved in system administration in a very visual way. All of the options available are presented in a clear and complete fashion.

Webmin proved to be an excellent tool for the purpose of this research. It allowed the researcher to keep a constant watch on the system changes while the honeynet was probed. It also provided the option of disabling any service, such as database, log files etc., remotely on the system, in case it is being exploited by the attackers. It was a utility which provided the researcher to keep a constant watch over the data collected from the honeynet. It also helped the researcher in administrating the systems, after the honeynet was probed or attacked, for the purpose of tracking file changes and backups. For screenshot see appendix A.5.

4.4 ACID (Analysis Console for Intrusion Databases)

Analysis Console for Intrusion Databases (ACID) is an analysis engine to search and process a database of security events generated by various intrusion detection systems. The Analysis Console for Intrusion Databases (ACID) is a PHP-based analysis engine to search and process a database of security events generated by various IDSes, firewalls, and network monitoring tools. The features currently include:

- **Query-builder and search interface** for finding alerts matching on alert meta information (e.g. signature, detection time) as well as the underlying network evidence (e.g. source/destination address, ports, payload, or flags).
- **Packet viewer (decoder)** will graphically display the layer-3 and layer-4 packet information of logged alerts
- **Alert management** by providing constructs to logically group alerts to create incidents (alert groups), deleting the handled alerts or false positives, exporting to email for collaboration, or archiving of alerts to transfer them between alert databases.
- **Chart and statistics generation** based on time, sensor, signature, protocol, IP address, TCP/UDP ports, or classification

ACID has the ability to analyse a wide variety of events which are post-processed into its database. Tools exist for the following formats:

- using Snort (www.snort.org)
 - Snort alerts
 - tcpdump binary logs
- using logsnorter (www.snort.org/downloads/logsnorter-0.2.tar.gz)
 - ipchains
 - iptables
 - ipfw

Due to its compatibility with Snort and MySQL, ACID was an appropriate analysis utility for the purpose of this research. ACID has a remarkable feature of classifying

the data collected according to their occurrences, which resulted in better and in-depth analysis of the data. It also had the compatibility of handling huge volume of data while analysing and generating graphs and charts. Therefore, it was an ideal utility to use for the analysis purpose of the data generated during this research. For screenshot see appendix A.1.

4.5 Syslog-ng

One of the most neglected areas of UNIX is handling system events. A daily check for system messages is crucial for the security and health conditions of a computer system. System logs contain many messages which are of no importance but at the same time they also contain many important messages which sometime get lost in the load of messages. Therefore, filtering is not always the best option. The better and consistent option would be having a logging service during runtime of all applications (Scheidler, 1999).

Syslog-ng, as the name shows, is a syslogd replacement, but with new functionality for the new generation. The original syslogd allows messages only to be sorted based on priority/facility pairs; syslog-ng adds the possibility to filter based on message contents using regular expressions. The new configuration scheme is intuitive and powerful. Forwarding logs over TCP and remembering all forwarding hops makes it ideal for firewall environments.

Syslog-ng is available free over the internet (<http://www.balabit.hu/en/downloads/syslog-ng/>) which may be redistributed or modified under the terms of the GNU General Public License as published by the Free Software Foundation.

Syslog-ng consists of message paths of one or more sources and also one or more destinations. Syslog-ng can be configured for logging on a remote location as well which can be useful in having a backup of logs. This method is very useful for establishment of honeypots as it allows storing the logs on a remote secure location other than storing on the Honeynet itself (which may get compromised).

In a honeypot, remote data storage is every essential to preserve the data from any fabrication or deletion. For the purpose of this research, Syslog-ng proved to be an excellent utility for transferring and storing the data generated in log files, while the honeynet was probed and attacked, to a remotely connected system. This provided a layer of security for preserving the data for this research.

4.6 Ethereal

Ethereal is a packet analyser (<http://www.ethereal.com>) released as Open Source under the GNU General Public Licence. Ethereal is available for almost every platform. For capturing the network traffic, libpcap is used which is available for nearly every known platform.

Ethereal has the feature of capturing live network traffic or to use a file as network traffic source (which has to be in tcpdump file format). All packet data is displayed in a readable format which makes it easy to analyse and is much simpler. It supports more than 50 types of protocols. Ethereal does decode known protocols as much as possible. For example, telnet traffic is shown in plain text.

Ethereal does also provide a feature of “Follow TCP Stream” which will present a special view to the user, where a chosen TCP stream is decoded and displayed. It does also provide the feature of filters both for display and capture packets. Limiting complex capturing to interesting packets is possible and flexible.

Ethereal does also provide a feature of “Protocol Hierarchy Statistic” which generates a hierarchical view of different packets captured. This provides a detail analysis of packets in terms of total percentage of each packet and also provides information of their start and end bytes.

Ethereal consumes a great amount of memory and affects the CPU performance. Depending on the packet sizes, it is a utility to decode and analyse packets which went over the wire.

The reason researcher decided in choosing the utility for the purpose of this research was purely because of its ability of analysing each individual packet separately. It was also capable of generating a protocol hierarchy statistic of the packets sniffed while the honeynet was probed. Ethereal was an excellent utility for analysing the malicious packets captured from the honeynet. For screenshot see appendix A.4

4.7 TCPDump

TCPDump (<http://www.tcpdump.org>) is a tool which can be used to capture and analyse network traffic. Its primary use is to record network traffic from a network device to a file. A Boolean expression may be used to identify certain packets with specific protocols, certain flags or packets destined to certain IP address. It also has the capability to get the packet stream out of a file. It is possible to simulate incoming packets by specifying a previously recorded session. This feature enables the user to record all network traffic having the possibility to analyse the network flow at a later time with the same tools which can be used in real time.

Snort is capable of writing the logs of all traffic in TCPDump readable format. Snort stores these logs in *tcpdump.log.xxx* file format. These files then can be read and analysed using Ethereal. These log files have been extensively analysed, in this project, using Ethereal.

4.8 Nmap

Nmap ("Network Mapper") (Fyodor, n.d) is an open source utility for network exploration or security auditing. It was designed to rapidly scan large networks, although it works equally well against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (ports) they are offering, what operating system (and OS version) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. Nmap runs on most types of computers, and both console and graphical versions are

available. Nmap is free software, available with full source code under the terms of the GNU GPL.

Following scanning techniques can be chosen by the user (Fyodor, n.d):

- UDP
- TCP connect and TCP SYN (half open)
- FTP proxy (bounce attack)
- Reverse-ident, ICMP (ping sweep), FIN, ACK sweep, Xmas Tree, SYN sweep, IP Protocol and Null scan.

After scanning one or more machines nmap shows a list of interesting ports on the target machines. It always gives the port's *open*, *filtered* or *unfiltered*. Open means that the target machine will accept a connection on that specific port number. Filtered means that a firewall or filter is protecting the port and nmap is unable to determine its state. Unfiltered means that the port is closed and there is no firewall or filter protecting the port.

The reason why Nmap was chosen was primarily because of its capability of scanning large networks using various scanning techniques. The information provided by the Nmap after the scan is completed is very useful in determining the features of the scanned hosts or network, such as what Operating system is installed and which ports are open with what services implemented on them. This allows more focused testing on those hosts by only using related exploits on them. For screenshot see appendix A.6.

4.9 GFI LANguard Network Security Scanner

GFI LANguard Network Security Scanner (N.S.S.) (<http://www.gfi.com.languard>) is a tool that allows network administrators to quickly and easily perform a network security audit. GFI LANguard N.S.S. combines the functions of a port scanner and a security scanner. It also creates reports that can be used to fix security issues on a network.

Unlike other security scanners, GFI LANguard N.S.S. does not create a "barrage" of information, which is virtually impossible to follow up. Rather, it helps highlight the most important information. It also provides hyperlinks to security sites to find out more about these vulnerabilities.

GFI LANguard N.S.S. is freeware for non-commercial use.

The researcher decided on using GFI LANguard as an other network scanning tool along with Nmap because of its ability of generating detailed report with details about various vulnerabilities outlined during network scan and also with the proposed solutions. Secondly, this provided the chance of triangulating the results generated by the Nmap to have the results more reliable by using multiple network scanning tools. For screenshot see appendix A.3.

4.10 Analyst Notebook 6

Analyst's Notebook 6, a commercial product developed by i2 Group (<http://www.i2group.com>), provides the optimum environment for effective link and timeline analysis. It is the de facto standard for this type of analysis worldwide and is an essential visualisation application. Analyst's Notebook helps in:

- Swiftly identify the common elements and links hidden within your data
- Easily explore and interpret complex information
- Apply a comprehensive range of analytical techniques to develop your intelligence
- Present the results of your analysis graphically, making them more accessible and easy to understand
- Produce dynamic briefing charts by including photos, video clips and documents
- Generate the analytical input that focuses an investigation and facilitates effective decision making and resource allocation.

The Analyst's Notebook produces charts in a range of formats, each applicable to a different technique. These typically include:

Link Analysis charts – present people, accounts, organisations and other entities together with the relationships or flow of commodities between them, no matter how complex the situation.

Network Analysis charts – extend the concept of link analysis to large datasets. These are usually generated automatically and are especially useful for identifying the key relationships in telephone, account and internet transactions.

Sequence of Events charts – reveal how related events unfold over time.

Transaction Pattern Analysis charts – depict when exchanges occur or commodities flow between entities.

Analyst proved to be excellent software for analysing the data generated in this research. It has an ability of converting huge amount of data into simple graphical pictures which are easy to understand and helps in analysing the various network communications between the hosts. The honeynet generated enormous amount of data which would not have been possible to analyse without using such utility. For screenshot see appendix A.2.

4.11 Conclusion

Honeypots gather a wide range of data from the environment and need a variety of tools for effective analysis. Various specialist tools were required for the purpose of the analysis of the collected data. An explanation of the various tools used for the analysing different formats of data collected has been outlined in the chapter.

The use of the correct tools for analysis is important as it affects the richness and reliability of the result.

In the next chapter, the researcher describes how the Operating System Fingerprinting tests were conducted for the purpose of the research.

5. Operating System (OS) Fingerprinting

5.1 Overview

There are various forms of information which a skilled hacker would like to gather before attacking any remote network or machine. One of the most important pieces of information is the type and version of operating system. With knowledge of the operating system, a hacker can look for any number of possible vulnerabilities that are specific to that operating system. There are various techniques by which information about the remote system can be identified. Apart from the classical techniques, discussed in Chapter 2, there are also various tools such as Nmap, Ettercap, Xprobe, which can be used in detecting the information about the remote operating system.

Honeyd, the virtual honeypot, is accompanied by a set of TCP/IP signature file (explained in section 2.2.6), provided by Nmap, of various operating systems, routers, switches, firewall, etc. It uses this signature file for emulating various hosts on to the network. Before implementing a honeypot on to the network, it was essential to determine how accurate each signature was. Therefore, an experiment was conducted to determine how many signatures specified in the Nmap fingerprint file (Fyodor, n.d) were successful in deceiving the attacker and the typical tools of trade.

5.2 Methodology

The purpose of this study was to investigate how different TCP/IP fingerprints could be detected by using network scanning tool, Nmap. A deceptive environment was setup on a machine (assume M1) using a freely available program called Honeyd. Honeyd is a small daemon that creates virtual hosts on a network. It is implemented as a Unix daemon that runs on a workstation and listens to network traffic. The hosts can be configured to run arbitrary services, and their TCP personality can be adapted so that they appear to be running certain versions of operating systems. Honeyd enables

a single host to claim multiple addresses. It was tested by setting up 3 different hosts (10.21.19.102-104).

It is possible to perform standard network diagnostics such as *ping* the virtual machines, or to *traceroute* them. Any type of service on the virtual machine can be potentially simulated. Instead of simulating a service, it is also possible to proxy it to another machine. Honeyd supports asymmetric routes and the integration of physical machines into the virtual network topology (Provos, 2002). Therefore, it is capable of re-routing the network traffic to any physical machine present on the network rather than simulating any service or operating system. The different TCP personalities are learned from reading an nmap fingerprint file. The configured personality is the operating system that nmap or xprobe will return. These personalities emulate the related operating system or network service but they do not exist in real. These are bogus services which can be emulated using their TCP/IP fingerprint. Personalities can be annotated to determine if they allow FIN-scans for open ports or to select the preference in which they reassemble fragmented IP packets.

Honeyd can be used to create a virtual honeynet or for general network monitoring. It supports the creation of a virtual network topology including dedicated routes and routers (routing configuration described earlier in section 2.3.4). The routes can be attributed with latency (the amount of time it takes a packet to travel from source to destination) and packet loss (packets which are lost or dropped in between while travelling from source to destination) to make the topology seem more realistic (*ibid*, 2002).

After the initial setup of Honeyd, arpd was configured. The ARP daemon moves the management of the ARP (Address Resolution Protocol) table from kernel to user space. It allows honeyd to use IP addresses on the existing network. Arpd can detect all the unallocated IP addresses on the existing network. It is useful for sites with large network segments (256+ systems per segment), because the kernel hash tables are not optimized to handle this situation (Provos, 2002). On successful configuration of both Honeyd and Arpd, a small configuration file was created which was mainly used to create multiple deceptive hosts running virtual operating systems.

5.3 Honeyd Configuration

Below is an example configuration of honeyd. This example defines a template which annotates a host running a web server.

```
annotate "AIX 4.0 - 4.2" fragment old
# Example of a simple host template and its binding
create template
set template personality "AIX 4.0 - 4.2"
add template tcp port 80 "sh scripts/web.sh"
add template tcp port 22 "sh scripts/test.sh $ipsrc $dport"
add template tcp port 23 proxy 10.23.1.2:23
set template default tcp action reset
bind 10.21.19.102 template
```

Virtual honeypots are configured via templates. A template is a reference for a completely configured computer system. New templates are created with the *create* command.

The *set* and *add* commands change the configuration of a template. Using *set* command we assign a fingerprint personality to the template. This personality determines the behaviour of the network as discussed earlier. The *set* command is also used to define the default behaviour of the supported network protocols. This default behaviour can be one of the following: *block*, *reset*, or *open*. *Block* means that all packets for the specified protocol are blocked by default, *reset* means that all ports are closed and *open* means that all ports are open by default for the specified protocol (Provos, 2003).

The *add* command is used to specify the services which can be accessed remotely. Beside the template name, we need to specify the protocol, port and the command to execute for each service. Honeyd also recognises the keyword *proxy* that allows us to forward network connections to a different host. The daemon expands the following four variables for both the service and the proxy statement: *\$ipsrc*, *\$ipdst*, *\$sport* and *\$dport*. This allows services to adapt their behaviour depending on the particular

network connection they are handling. It is also possible to direct the network traffic back to the host who is probing the honeypot (*ibid*, 2003).

The *bind* command is used to assign a template to an IP address. If no template has been assigned to an IP address, the *default* template is used. The above mentioned configuration example creates a template for a personality “AIX 4.0 -4.2” and adds http, telnet and ssh ports. Finally it binds all this information to a virtual IP address 10.21.19.102. Similarly there were 2 more hosts were created.

After creating the configuration file, arpd was processed this was followed by honeyd. This allowed arpd to respond to any incoming network traffic directed towards the honeyd. If the incoming IP address request does not exist in the configured honeyd configuration file, arpd simply drops that packet and processes another request. Thereafter, from a different machine (say M2), nmap was used to carry out the scan on those virtual hosts created in the configuration file. About 458 operating systems signatures were used to scan from the fingerprint database file called *nmap.prints*. Since the task was to find out the details of operating systems running on the virtual daemon – honeyd, nmap was used to carry out SYN Stealth attack with fast scanning option along with Don't ping and OS detection options. The representation was

```
nmap -sS -F -O -PO <ip address>
```

Scan Types

-sS

TCP SYN scan: This technique is often referred to as “half-open” scanning. A SYN packet is send as if it is going to open a real connection and waits for the response. A SYN|ACK indicates that the port is listening. The primary advantage of this scanning technique is that fewer sites will log it.

-F

Fast Scan Mode: This specifies that the user only wishes to scan for ports listed in the services file which comes with nmap or the configuration file created. This is obviously much faster than scanning all 65535 ports on a host.

-PO

Don't Ping: Do not try and ping hosts at all before scanning them. This allows the scanning of networks that don't allow ICMP echo requests (or responses) through their firewall.

-O

This option activates remote host identification via TCP/IP fingerprinting. In other words, it uses a bunch of techniques to detect subtleties in the underlying operating system network stack of the computers which are scanned. It uses this information to create a fingerprint which it compares with its database of known OS fingerprints (the *nmap.prints* file) to decide what type of system is getting scanned.

The -O option also enables several other tests. One is the "Uptime" measurement, which uses the TCP timestamp option to guess when a machine was last rebooted. This is only reported for machines which provide this information.

5.4 Test Results

Total No. of Fingerprints in nmap database	458
No. of Detected by nmap	231
No. of Undetected by nmap	227
Percentage Detected	50.43%
No. of Parsing Error Configuration	5
No. of 'NO' results (i.e. nmap kept scanning)	9
No. of Routers/Switches/Hubs signatures in nmap fingerprint database	70 (approx.)
No. of Routers/Switches/Hubs detected by nmap	39 (approx.)
Percentage Detected	55.71% (appx.)
No. of Printers/PrintServers signatures in nmap fingerprint database	25 (approx.)
No. of Printers/PrintServers detected by nmap	14 (approx.)
Percentage Detected	56.0% (appx.)
No. of Firewalls signatures in namp fingerprint database	14 (approx.)
No. of Firewalls detected by nmap	6 (approx.)
Percentage Detected	42.85% (appx.)

Table 5.1: Nmap Scanning Results

5.5 Discussion

The purpose of this investigation was to determine how many signatures of operating systems, routers, switches, hubs or printers listed in nmap signature file can be actually detected by nmap scanner. There were 458 signatures listed in the nmap signature file. As the above results indicate only 231 signatures out of 458 were actually detected by the nmap scanner. Only 50.43 percent signatures can be traced by nmap scanner. There were 5 signatures in the signature database which were wrongly parsed and about 9 signatures did not give any results. Nmap scanner just continued scanning those 9 signatures for a long time and there were no results.

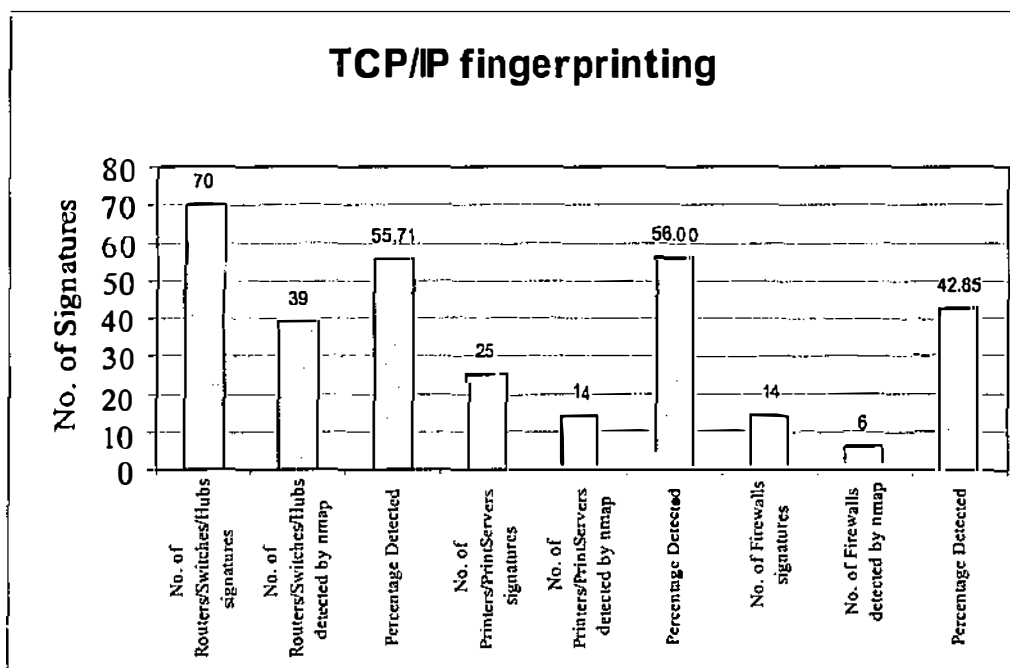


Figure 5.1: TCP/IP Fingerprinting Results using Nmap

The signature database file had approximately 70 signatures of Routers/switches/hubs out of which only 39 were actually detected by nmap scanner. There were also approximately 25 signatures of Printers/Print servers from which only 14 signatures were detected. Out of approximately 14 signatures of firewalls, only 6 signatures were detected by the nmap scanner. As only 42.85 per cent of the firewall signatures were

detected by nmap, it may be justified to say that nmap is not very effective in determining the Firewalls on the network.

5.6 Conclusion

Honeyd as a defensive tool has a wide application, due to the fact that over 50% of the signatures are usable in deploying a virtual honeypot network. By using the tested and validated signatures it will allow any competent systems administrator to provide front-line defensive deception honeypot network that will provide valuable attack intelligence. Using these results the reliability of deception has increased as only validated signatures can only be used for honeynet implementation. This provides an advantage of maintaining a higher level of deception as we can now eliminate the use of bogus signatures which otherwise could have identified the presence of deception and made the attacker suspicious about the nature of the network.

Based on the deception created in the experiment, it can be classified as Mimicry. The researcher created the replica of different types of hosts using the features of real-time hosts using Honeyd. These hosts did not exist in reality but were virtually created using their fingerprint features provided with Honeyd. Therefore, these hosts were mimicked to pretend as real hosts with few characteristics of real hosts such as running web servers, services like telnet, ftp, etc.

All the unsuccessful fingerprints could now be filtered from the signature file and hence only the accurate signatures could be used for creating a successful deception using Honeyd.

The next phase of this research was developing honeypot systems based on the validated signatures and testing these configurations for the ability to deceive operating system fingerprinting scanners and other network attack tools.

6. The Honeypot Implementation

After discussing what a honeypot is, its value and advantages & disadvantages in chapter 2, the next step is to identify which honeypot will be effective for our purpose. Incorrectly selected or implemented honeypots can be dangerous as they may be compromised easily by attackers. These compromised honeypots then may be used by attackers to carry out attacks on other hosts on different networks.

Before selecting any honeypot it is important to identify the goals for which the honeypot is required and then selecting the appropriate honeypot to be implemented.

6.1 Goals

Honeypot solutions can be implemented for various purposes such as Spitzner (2002) states:

- Preventing attacks through deception or deterrence.
- Detecting attacks.
- Responding to attacks, collecting data and evidence of attacker's activities.
- Researching attackers' tools, tactics and moves.

To determine which of the above purpose would fulfil our requirement, the honeypots can be categorised into two categories: production and research. The production honeypots are used for the protection of any resource or organisation. On the other hand, research honeypots are used for gathering information on attackers.

Production honeypots help secure organizations in one of three ways—preventing, detecting, or helping them respond to attacks while research honeypots helps in learning about various attackers and their tools and tactics. Production honeypots are generally highly secured in compare to research honeypots. They are normally used by organisations as a layer of security architecture to protect their real legitimate network. They are used for providing deterrence. On the other hand, research

honeypots are used as a learning tool. They are used to gather maximum amount of attack information, to understand and learn from various tools and tactics used by attackers while probing and attacking the network. This attack intelligence is used to harden the actual legitimate networks.

A research honeypot provides extensive value by information gathering (Spitzner, 2003). They can be used to capture automated threats, malicious programs or scripts run by malicious attackers. By analysing this captured data it helps in by improving the security of other legitimate networks. It allows us to learn more about advance attacks.

The significance of this research was to improve the defensive capability of Honeynet networks. This research is significant to:

1. Universities, government organisations, educational institutions who will be able to use the data generated from this research as a platform to continue with further research into computer and network security.
2. Private organisations or individuals can use this research for deploying this network architecture into their networks with the aim of making it more secure.

Therefore, the main purpose of carrying out this research was:

1. To improve the level of Deception presented to attackers in Honeypot design.
2. To harden a deceptive honeypot and test its effectiveness using an empirical learning approach.
3. To improve the ability of a deceptive honeypot to gather attack intelligence.

On the basis of above mentioned significance and purpose of research, the researcher's purpose was to gather information about the attack intelligence in a Honeynet environment. Therefore, a research honeypot was appropriate for this research. Once the goal of the honeypot was identified, to gather attack information, the next step was to determine which honeypot was to be used for information gathering (Spitzner, 2003).

6.2 Selecting a Honeypot

Once the goal is clearly defined, next step is to select the honeypot solution. Every honeypot has its own strengths and weaknesses. When selecting any honeypot, following criteria may be considered (*ibid*, 2002):

- *Level of interaction.* The level of interaction defines the type of functionality the honeypot provides. The greater the interaction, the more can be learned from the honeypot but it too increases the complexity of design and risk of corporate network.
- *Commercial versus homemade or freeware.* Commercial honeypots are easy to configure, manage and support but some of them cost more for such functionality. Homemade or freeware can be customised solutions but may not be effective a permanent basis on the organisational network.
- *Platform.* It is important to determine on which platform the honeypot should run. The baseline operating system impacts the performance of the honeypot and also how the organisation can manage it.

6.2.1 Interaction Level

Defining the level of interaction is a very critical issue when selecting the honeypot type. The greater the interaction, the more an attacker can do and the higher probability that more can be learned from it. However, the greater the interaction, the more functionality is provided to the attacker, the greater the complexity of configuration. With the high interaction honeypots attacker have more functionality in terms of real time operating systems, real services to probe and carry out attacks. But the more the attacker is allowed to do, the more is the risk of honeypot becoming

compromised and used to attack other systems. There are 3 main type of level of interactions as defined by Spitzner (2003), these are:

Low-Interaction Honeypots

These are typically easy to install, configure, deploy and manage because they are simple in design and have basic functionality such as they may simulate simple logins but nothing more which an attacker could do. In a low-level interaction, the attacker is limited to interact with the predesignated services such as FTP, Telnet etc.. For example, a low level interaction honeypot can emulate a standard UNIX server with simple service like Telnet. Attackers could Telnet to the honeypot, and probably obtain a login prompt. They could carry out a passwords brute force attack to gain unauthorised login but there may not be a real operating system for them to log on to.

The primary value of low-level interaction honeypots is detection of any unauthorised activity. Low-interaction honeypots has the limited functionality that they can only emulate some services. Because they are simple in nature, they have the lowest level of risk. These are easy to deploy and maintain because they have limited interaction capabilities, which also reduces risk.

Medium-Interaction Honeypots

Medium-interaction honeypots offer attackers a greater ability to interact with the honeypot. They are expected to give higher performance than the low-interaction honeypots. For example, in a real operating system, a partition may be created to create a virtual operating system within the real operating system. The virtual operating system can be controlled by the real operating system but will give the feel of a real operating system to the attacker. Continuing with the Telnet example, in a medium interaction honeypot, when an attacker tries to connect to the real network using telnet, they would be allowed a successful login with a basic level of interaction with the emulated file system. This will allow an attacker to interact with the file system functionality as if it is a real file system. This level of interaction is greater than the low level honeypot, which would have most likely presented a successful login banner and nothing else.

High-Interaction Honeypots

High-interaction honeypots are at the highest level of honeypot technology. They provide the user with a vast amount of information about attackers but also they are highly time consuming and often difficult to maintain. High-interaction honeypots does not emulate any operating system or service they instead present a real system to the attackers. A high-interaction honeypot does not emulate FTP or Web services—instead, it installs and uses a real FTP (such as wu-ftp) or Web server (such as Microsoft's IIS).

High-interaction honeypots are placed in a very highly controlled network environment such as behind the firewalls and other security perimeters. The ability to control the attacker's movement is not controlled directly by the honeypot but is done through the use of perimeter countermeasures such as a firewall. The firewall allows the attacker to probe or attack the honeypot but does not allow them to carry out attacks using that honeypot once it is compromised. Such network architectures are difficult to maintain, monitor and manage therefore, high-interaction honeypots are difficult to configure and maintain.

Conclusion

With regard to this research, low interaction honeypots would have not been very useful for data collection. Low interaction honeypots do not allow high level of interaction between the emulated services and attackers. They are also not capable of emulating enough services. For research based honeypots, the more the interaction the better, to enable the gathering of attack intelligence. Therefore, low-interaction honeypots do not provide much support for the purpose of this research.

However, medium-interaction honeypots are increasingly complex and so start to possess a higher risk profile for compromise or discovery. The more realistic an emulated service or operating system is given to the attacker, the more easier it may become for an attacker to break the virtual environment and take control of the real operating system. The higher functionality and complexity may make it easier for an attacker to compromise it. These honeypots are more time consuming to install and configure than low level honeypots. Deploying and maintaining medium-interaction

honeypots are more complicated than low-level honeypots therefore require higher level of security. However these honeypots can gather a higher amount of information than low-level and thus provides us with higher level of intelligence.

Since the purpose of this honeypot was to gather information about the attackers movements and their activities onto the network, a medium-level of interaction honeypot was a viable solution. It provided the basic functionality required by the researcher such as emulated services, extensive data collection and high performance in compare to low-interaction honeypots. High interaction honeypots were not feasible for the purpose of this research. As the researcher had to deploy a research honeypot and high interaction honeypots are more suitable for production honeypots, therefore medium-level interaction honeypots were more suitable for the purpose of this research.

Table 6.1 summarises the above mentioned level of interactions (Adopted from Spitzner, 2003, p77):

Level of Interaction	To Install and Configure	To deploy and Maintain	Information Gathering	Level of Risk	Performance
Low	Easy	Easy	Limited	Low	Low
Medium	Involved	Involved	Variable	Medium	Medium
High	Difficult	Difficult	Extensive	High	Medium-High

Table 6.1: Honeypot Level of Interaction

6.2.2 Commercial versus Homemade or freeware Solutions

A commercial honeypot is easier to install, configure, deploy and maintain. Most of the commercial honeypots have a GUI interface, making it easier for the user to understand the technology. Commercial honeypots also provide vendor based support services to the end user. There is often a considerable amount of documentation and training available from the developer of commercial honeypots which assists the installation, configuration and maintenance of the honeypot.

Homemade honeypots are developed by the users according to their need. It depends on the resources and technology the user possess. Homemade honeypots can be used for various purposes. They can be designed and developed according to the level of interaction required. For example, if the user wants to detect the certain types of port scans then he/she need nothing more than a simple program that emulates that particular port or different ports and captures all the activity to that port. Alternatively, homemade honeypots can be developed to emulate the whole operating system, which may allow the attackers to attack the host using various tools and tactics. This will provide a higher level of interaction and will help in gathering more attack intelligence. Therefore, development of these types of honeypots purely depends on the requirements and technical knowledge of the user (*ibid*, 2003).

Homemade or freeware honeypots are cheaper when compared to commercial honeypots. Homemade honeypots may be customised according to the users' requirements and thus provides better flexibility in use. Homemade honeypots can be updated frequently while with commercial honeypots, the user has to wait for the developer to provide with the update feature (*ibid*, 2003).

For the purpose of this research, developing a homemade honeypot from ground up was not a viable solution as it would have taken lot of time and resources to develop. As the honeypot suppose to be a research honeypot a commercial honeypot was also not the best solution because of financial restriction to the researcher and the lack of customisation. Therefore, a freeware honeypot was best suited to the task. Moreover, an open source honeypot was more ideal for the research as it allowed the researcher to configure the honeypot according to the need and requirement of the research. Open source honeypot was more flexible to deploy as the researcher did not had to depend on vendor for support and maintenance.

6.2.3 Operating/System Platform

The third criterion is selecting a platform for the honeypot. Several commercial honeypots are based on Windows operating system which generally makes them easier to install, configure and maintain when compared to command-line based

operating systems such as UNIX. UNIX being an open source operating system, allows the user to configure and modify the files according to the requirements of the honeypot. Windows is a commercial operating system so a limited level of modification and customisation is possible.

Conclusion

Since the researcher decided to choose a freeware and open source honeypot, so there was little choice in determining the platform. Open source programs are mainly available in UNIX platform, so the researcher decided to implement the honeypot on a UNIX platform.

On reviewing the above options, Honeyd (Provos, 2002) was the only honeypot which suited the requirements for this research. Honeyd, an open source honeypot, is a medium level honeypot which is implemented on UNIX platform. Also according to the Figure 2.13, p57, honeyd has medium level complexity and high performance in detecting and gathering attack intelligence.

6.3 Determining the number and location of Honeypots

It is essential to determine the location of the deployment of a honeypot on any network. Most of the production honeypots are placed behind the organisation's security perimeter. Production honeypots are mainly used for protection and detection; therefore if any malicious attack has passed through the initial security perimeter of the organisation such as firewall then it is necessary to detect such attacks within the network before they do anything malicious within the network. Such honeypots are placed where there is high degree of risks involved such as internal network, DMZ. These honeypots are always behind the firewall perimeter thus helps in determining if anything has passed the firewall security (Spitzner, 2003).

Research honeypots are used to gain information on threats. If the researcher is concerned with the internal threats then the honeypot may be deployed in the internal network of the organisation. Research honeypots may also be deployed outside the

firewall perimeter as well. This will ensure that honeypot is open to potential attacks. But this possesses a great risk too. If the honeypot is compromised at this location then it may be easier for the attacker to attack various systems using the organisation as a launching platform to carry out the various attacks

This research was artificial in nature as it was conducted in a controlled laboratory environment. The honeypot was implemented inside a controlled laboratory network which did not have any connection to any other network.

After determining the location and type of honeypot required, it was important to know how many of those honeypots were required to implement in the network architecture. The number of honeypots depends on the size of the network as well as on the available resources. For large organisations with various networks, one or two honeypots may not be sufficient for effective deployment.

The goals specified for the honeypots plays an important role in determining the number of honeypots. For production honeypots, several honeypots may be required to secure the environment. For research honeypots, one or two honeypots may be deployed on different location of the network (Spitzner, 2002). Deploying numerous research honeypots has limited value as it is more likely to gain same information with only one or two honeypots, while having various production honeypots provides with greater value. For example, if the honeypot has been deployed on the organisation external network to detect attacks, it will not be able to detect any internal attacks. Therefore, for full network attack detection purpose, the honeypots should be deployed on various location of the network.

However, it should be noted that honeypots do not solve any security problem; they only contribute to the overall security architecture. Therefore, enough honeypots should be selected which could contribute to organisation security but should not consume lot of extra resources which could have been used by other security mechanism.

After deciding on the type of honeypot required for data gathering, the next step was to identify the number of honeypots required and the location for deploying it. Since it

was a closed laboratory experiment for research purpose therefore a single honeypot was sufficient to gather attack information within a closed laboratory perimeter. The figure 6.1 below illustrates the laboratory network structure with the honeypot and a sniffer deployment for data gathering purpose:

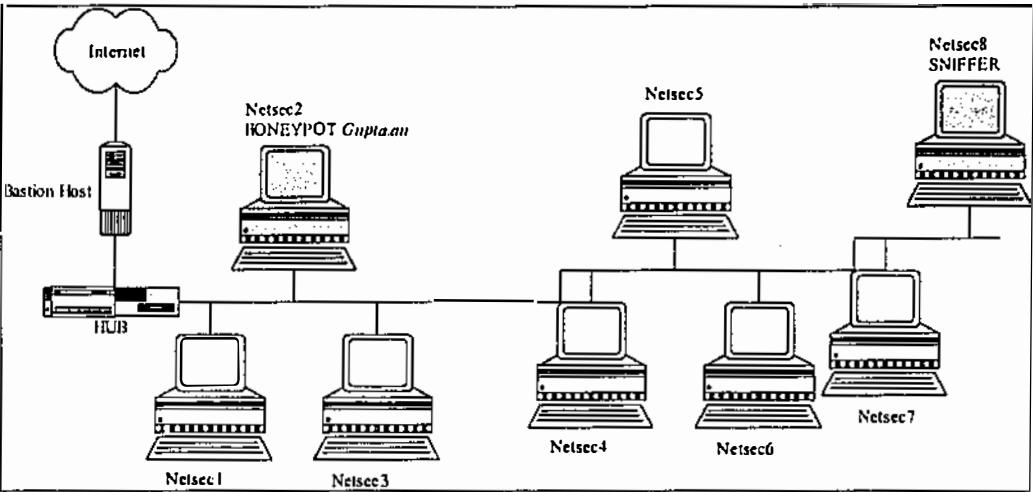


Figure 6.1: Deployment of Honeypot in a Laboratory Network Structure

In the above network, there are 8 hosts (named Netsec1...8) which are connected to a hub. A hub was used because it was possible to sniff the network traffic as all the ports were on the same collision domain whereas on switch sniffing is not possible as each port is on a different collision domain. The hub is connected to a bastion host which connects to the Internet. The bastion host acts as a server for the network connected to the hub and also acts as a firewall to protect the network from external attacks. Host *Netsec2* was used to deploy a Honeyd honeypot, *Gupta.au*, and *Netsec8* was used to install a sniffer Snort to gather attack information over the network. *Gupta.au* consisted of RedHat Linux 7.3 running on a computer with following configuration:

CPU	Pentium 3, 450MHz
Memory	256Mb
Hard disk	6 GB
Operating System	RedHat Linux 7.3
Installed Softwares	HoneyD 0.4, Syslog-ng

Table 6.2: The Honeypot System Configuration

In the above configuration, only one interface card was used for the honeypot implementation.

The Honeyd honeypot was configured with unused IP address to create a virtual Honeynet network. The figure 6.2 below illustrates the virtual network structure created within the Honeyd honeypot:

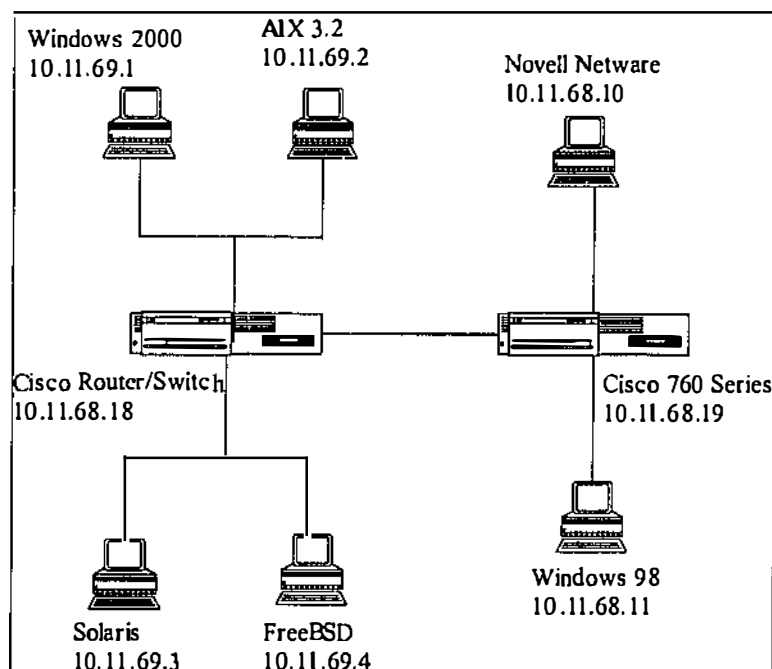


Figure 6.2: The Virtual Honeynet

Honeyd has a feature of creating virtual IP addresses and services. There were two internal networks created (10.11.68.0/24 and 10.11.69.0/24) both connected to each other using Cisco routers. The table 6.3 below summarises the allocated IP addresses and services on the Honeyd virtual network:

IP Addresses	Signatures	Services
10.11.69.1	Windows 2000 Professional, Build 2218	Http – port 80
10.11.69.2	AIX 3.2	Http – port 80
10.11.69.3	Solaris 2.3 – 2.4	Http – port 80
10.11.69.4	FreeBSD 3.2 – 4.0	Http – port 80
10.11.68.10	Novell Netware 3.12 or 386 TCP/IP	Http – port 80
10.11.68.11	Windows 98	Http – port 80
10.11.68.18	Cisco Router/Switch with IOS 11.2	Http – port 80
10.11.68.19	Cisco 760 Series (non IOS) or IBM Stackable Hub	Http – port 80
<i>Default</i>	Windows 98	Http – port 80 Netbios – port 139

Table 6.3: Summary of IP addresses allocated on corporate network

The 10.11.69.0/24 network was running Windows 2000 Professional server (10.11.69.1), AIX 3.2 server (10.11.69.2), Solaris 2.3-2.4 server (10.11.69.4), Free BSD 3.2 – 4.0 server (10.11.69.4) and Windows 98 (as default) on rest of the remaining hosts of the 10.11.69.0/24 network. Hosts with IP address from 10.11.69.1 – 10.11.69.4 were running only *http* (port 80) service with a default script (filename *web.sh*). The *http* service was emulated on the network because it has been always an attractive target for the hackers. Using the http port, hackers may be able to compromise the web-servers which may eventually give them access to the operating system and also if the http port is compromised then the attackers can defaced the websites which may have a lot of impact on the organisation depending on the nature of the organisation. All the other remaining hosts of the 10.11.69.0/24 network were running Windows 98 as default with services open on port 80 (http) and port 139 (netbios). The reason Windows 98 was setup as a default operating system on the network was to resemble the network as much as possible close to the architecture of corporate network.

The 10.11.68.0/24 network was connected to the 10.11.69.0/24 network using Cisco Router/Switch with IOS 11.2 (IP address 10.11.68.18) which in turn is connected to another Cisco 760 Series (non IOS) or IBM Stackable Hub (IP address 10.11.68.19).

The 10.11.68.0/24 network was running Novell Netware 3.12 or 386 TCP/IP server (IP address 10.11.68.10), Windows 98 (IP address 10.11.68.11). Also Windows 98 was implemented as default on all the other remaining clients of the network.

This complete network structure was configured over the RedHat Linux 7.3 using Honeyd 0.4. For configuration file (honeyd.conf) refer to Appendix C.I.

Data Collection was the main aspect of a honeypot and to store the data collected securely. Therefore, a separate machine was setup for sniffing and storing the data. This machine consisted of a RedHat Linux 7.3 running with following configuration:

CPU	Pentium 3, 450MHz
Memory	256Mb
Hard disk	6 GB
Operating System	RedHat Linux 7.3
Installed Softwares	MySQL, ACID, Apache, Webmin, Snort, Syslog-ng

Table 6.4: Data Collection Machine Specification

Snort IDS was used as a sniffer to collect information about all inbound and outbound traffic over the network. Snort seemed to be a good and cheap solution for this purpose as it is freely available over the internet. Snort also has the capability of connecting to a MySQL database. All the data collected by snort was transferred to the MySQL database. The data stored in the database was used by ACID (Analysis Control of Intrusion Databases) for making a complete analysis by generating graphs and charts using the data stored in the database using Snort.

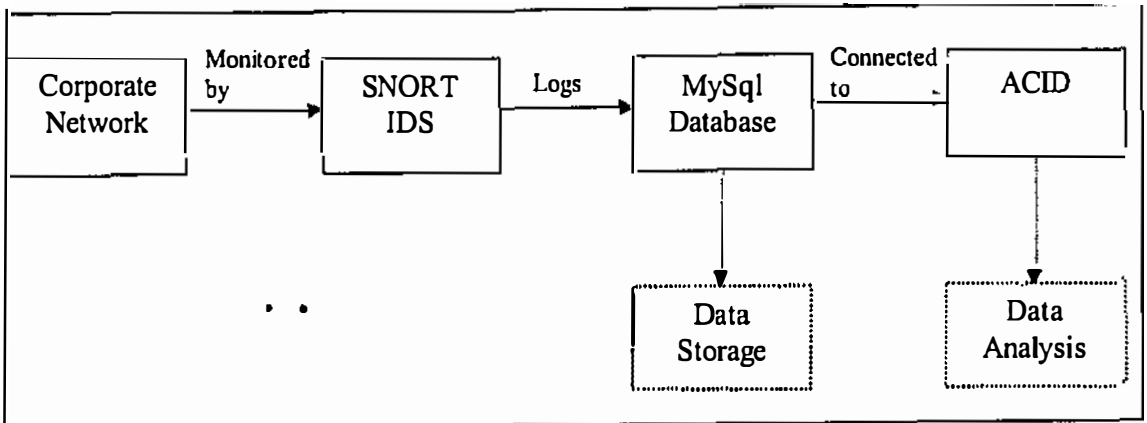


Figure 6.3: Architecture of Data Collection Machine

Another set of data was also logged using syslog-ng. This data was also stored remotely on a different machine, so that in the event that the data collection machine were to be compromised then there is another set of data available as a backup. Webmin was used for administering the honeypot remotely. As the researcher was more interested in the tools and programs or scripts used by the attackers during the network penetration exercise, therefore collecting the hardware information of the attackers' machines was irrelevant for the purpose of this research.

6.4 Conclusion

A Honeypot is implemented based on the goals and level of interaction required for the purpose of the research. This chapter described the various issues such as goals, level of interaction, the operating platform which is necessary to consider before implementing the honeypot.

Due to restrictions imposed by resourcing and the research model and addressing these issues relating to honeypot design, it was determined that only single honeypot would be deployed for the research. The design though singular fulfilled research requirements and was able to accomplish with existing resources. The honeypot design was constructed to allow complete and accurate collection of attack intelligence that could be gathered from attackers' activities. While the use of multiple honeypots would be an ideal situation it is beyond the scope and resources of this thesis.

7. First Test results on Honeyd 0.4a

7.1 Overview

During the initial testing phase, a group of students were asked to probe the designed network (10.11.68.0/24 and 10.11.69.0/24, Figure 6.2, pl 10). These students were asked to take their places in one single laboratory. This was mainly done because only one laboratory was available. All the selected students brought their own laptops and used their own choice of programs and tools for conducting the exercise. This testing phase existed between the time window of 17-02-2003 [11:0028] to 18-02-2003 [17:58:01]. There were various attacks attempted by students which generated a substantial amount of data on the machines monitoring the network. The intrusion data generated during this period was stored on a database which was used by ACID (Analysis Console for Intrusion Databases) for the analysis purpose. There were also various log files such as syslog – logs generated by Linux operating system for all the activity associated with the machine was analysed manually, tcpdump log file which is a raw dump of network traffic was analysed using ethereal, and temporary system logs. A few of the attackers also provided feedback and reports which assisted in analysing the data.

7.2 ACID Analysis

From the initial testing the Snort IDS logged and recorded 19451 alerts into an SQL database. The format of the SQL database (see Appendix C.7 for sql database script) confirms that there are 498 unique alerts which were divided into 11 different categories.

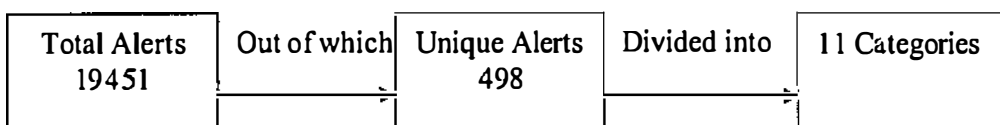


Figure 7.1 Acid Analysis

Classification	Total Alerts	Signatures
Unclassified	1032 (5%)	285
Misc-activity	494 (3%)	3
Bad-unknown	5764 (30%)	2
Attempted-recon	8369 (43%)	79
Web-application activity	2957 (15%)	90
Web-application attack	748 (4%)	30
Misc-attack	7 (0%)	1
Attempted-dos	56 (0%)	5
Protocol-command-decode	16 (0%)	1
Attempted users	3 (0%)	1
Successful admin	5 (0%)	1

Table 7.1: 11 Alert categories

The above classified alerts mentioned in table are described in detail below:

Unclassified

There were 285 signatures recorded under this classification. These signatures were different port scans on various hosts and servers of the network. To identify various exploits and in an attempt to fingerprint the network, attackers would perform portscans. This would have provided them with details of various open and closed TCP and UDP ports e.g. 21-FTP, 22-SSH, 23-Telnet, 25-SMTP, 80-HTTP. This activity would indicate that the attackers tried retrieving information about various hosts and servers like what operating system they are operating on and what port numbers are available for remote access to gain access into the network.

Misc-activity

There were only 3 signatures which were identified under misc-activity. These signatures were:

- *Bad traffic udp port 0 traffic (Nessus). (CVE 1999-0675)*

This is associated with Checkpoint Firewall-1 vulnerability. CheckPoint Firewall-1 can be subjected to denial of service (DoS) via UDP packets that are sent through VPN-1 to port 0 of a host. It seems that Nessus (a network probing tool) tried to send large amount of udp traffic at port 0 to attempt a denial of service attack. This attack would have disabled the CheckPoint

Firewall-1 (if exist on the network) and all the incoming and outgoing network traffic would have passed through without any check. Such attack could facilitate full access to the network and could be extremely dangerous for the entire network. It was identified that this attack originated from 2 source IP addresses and was destined to 135 different hosts on the network. Out of the 2 source IP address, one of them was of the hacker (172.16.1.200) and the other IP address was of a host (10.21.19.5) within the network. This could have been as a result of traffic being redirected using the victim host to hide the origin of the real attack.

This attack attempt indicates that the attacker tried causing a Denial of Service attack on the CheckPoint firewall by sending large amount of UDP packets. This could have disabled the firewall and all network traffic would have entered into the network without any rule check specified by firewall.

- *BAD TRAFFIC bad frag bits*

This is similar to the previously discussed signature. The attacker sends large amount of UDP packets to flood the hosts or servers to attempt a denial of service attack. This indicates that the attackers tried to disable the servers and hosts from processing any network request and disrupt the normal services of those hosts and servers.

- *EXPERIMENTAL MISC AFS access (Nessus)*

AFS is a distributed file system that enables sharing files across both local area and wide area networks. There were miscellaneous attempts of accessing *AFS* by attackers using Nessus. If such attempt is successful, then the attacker can have full access to the filesystem of the hosts and can easily cause damage to the system. It was identified that this attack originated from 2 sources and was destined to 21 different hosts. The source IP addresses were the same as one of them was of the attacker's (172.16.1.200) and the other was of the victim (10.21.19.5) on the network. Therefore, it is reasonable to conclude that host 10.21.19.5 was one of the compromised host which the attacker was using to attempt his or her attacks.

The above discussed attack indicates that the attacker tried gaining access of the file system to have access on the system. He or she also used one of the compromised hosts to achieve its goal. If the attacker would have succeeded in gaining the full access to the file system then it may have compromised the whole honeynet and probably the system could be used as the launching pad for attacks on other networks.

Bad-unknown

There were 2 signatures which were identified under this category. They were:

- *ICMP Redirect host*

This comprised of 30% of the total number of alerts recorded over the network. This may have occurred because all the network traffic was routed through a gateway machine within the laboratory which further forwarded the network traffic to their respective destinations. However it cannot be denied that some of the attackers would have used this method as part of their attacking tactics too. This type of attack is capable of crashing or locking up a host machine.

Since the total number of ICMP alerts was very high, it appears that the attackers tried flooding few hosts with ICMP messages using ICMP redirect. This could have caused the hosts over flooded with ICMP requests and therefore those hosts may not be able to respond to other network requests. But since the researcher was routing all the network traffic through a gateway machine it was not possible for him to block or drop such ICMP requests on the network.

- *MISC Large UDP packets*

Large UDP packets were used by the attackers to flood some of the hosts with UDP request to attempt a DoS type of attack on the particular hosts and servers. This could be mainly to bring down the various servers and hosts on the network which eventually would have collapsed the whole network.

Attempted-recon

There were 79 different signatures which were identified in this category. Some of the most prominent were:

- *SCAN SOCKS Proxy attempt*

SOCKS is an IETF (Internet Engineering Task Force) approved standard (RFC 1928) generic, proxy protocol for TCP/IP-based networking applications. The SOCKS protocol provides a flexible framework for developing secure communications by easily integrating other security technologies. When an application client needs to connect to an application server, the client connects to a SOCKS proxy server. The proxy server connects to the application server on behalf of the client, and relays data between the client and the application server. For the application server, the proxy server is the client.

It was identified that there were SCAN SOCKS Proxy attempt on 183 destination hosts. It appears that some of the attackers tried to probe the SOCKS proxy to connect to various servers available on the network.

- *SNMP Request UDP (CAN 2002-0012)*

Vulnerabilities in a large number of SNMP (Simple Network Management Protocol) implementations allow remote attackers to cause a denial of service or gain privileges via SNMPv1 trap handling.

From the collected data it is believed that there were large amount of SNMP requests (8% of alerts) were broadcasted on various hosts with the intention of flooding those hosts with SNMP Request. It was identified that about 156 hosts were subject to this attack by various attackers. This could have caused DoS attack on those hosts and would have made them unavailable for other requests.

- *ICMP Ping NMAP*

It appears that Nmap (a network mapping tool) was used in sending ping requests to the network to check whether a host is responsive or alive. This is one of the basic starting strategies used before attempting any attacks on any host or network.

- Large number of *WEB-CGI* and *WEB-MISC* signatures. This illustrates that lot of attempts were made on web based services and ports. This indicates that web-based attacks were more common among attackers to exploit. Also the network configuration too contained large number of hosts with only *http port 80* active on them, so web-based attacks were expected by the research in high number.

Web-application activity and Web-application attack

There were 90 and 30 unique signatures in both categories respectively. These signatures were focused on either IIS Script access based attacks or CGI and MISC web based attacks. On the basis of the data collected, it appears that web based attacks were the most common form of attacks used by attackers. There were more than 120 signatures identified which were used for attacking web based services and port numbers. Therefore, the researcher believes that the web based services are prone to malicious attacks and hence have a high rate of scanning.

Misc-attack

There was only one signature identified under this category which was subject to WEB-PHP directory.php access (CAN 2002-0434). It allows remote attackers to execute arbitrary commands via shell meta characters in the *dir* parameter. It indicates that the PHP based attacks were not so attractive for the attackers to probe. There was hardly any type of probe or activity which tried to exploit any PHP vulnerability.

Attempted-dos

This category included 5 alert signatures each related to distributed denial of service attack. These attacks were classified under CAN 2000-0138 for CVE. It states that “a system has DDOS attack master, agent or zombie installed such as 1) Trin00, 2) Tribe

Flood Network (TFN), 3) TFN2K, 4) stacheldraft 5) mstream, and 6) shaft." This attack was destined to 10 different hosts. It indicates that the attackers were very keen on implementing distributed denial of service attacks on the network as previously there were other occasions too where attackers tried to carry out DoS attacks on various hosts and servers. If these attacks were successful, they may have brought down the whole network and would have made it inaccessible for any user.

Protocol command decode

This was a single MISC IPSEC PGPNet connection attempt to connect to IPSec. It indicates that attacks associated with the PGP (Pretty Good Privacy) were not too popular among the attackers or they probably lagged the skills of conducting such attacks.

Attempted users

This was a single alert to access the WEB CGI webspeed. The webspeed configuration program does not properly disable access to the WSMAdmin utility, which allow remote attackers to gain privileges.

Successful admin

This was a single attempt of retrieving Trivial File Transfer Protocol (TFTP) GET password. It uses the User Datagram Protocol (UDP) and provides no security features. It is often used by servers to boot diskless workstations, X-terminals, and routers. This indicates that there was an attempt to gain administrative access to the host or server by retrieving its password using TFTP GET command and thereafter booting the terminals to gain full access.

Conclusion

From the above discussion it is clear that attackers tried various types of attacks. Of these attacks the most prominent were Web based attacks and probes. It seems that the attackers were more inclined towards attempting web-based vulnerabilities. There were also large numbers of port scans on various hosts. There was also evidence which shows that the attackers tried to conduct some DDoS attacks on the network. Some of the attackers tried flooding the hosts with large number of SNMP requests to carry out the DoS attacks on the servers. From the analysed data, it appears that the

attackers were more curious to attempt either DoS based attacks or exploit web based vulnerabilities. Out of all the above discussed alerts, the most frequent 5 alerts were:

Signature	Classification	Total
ICMP redirect host	Bad-unknown	5743 (30%)
SNMP request udp	Attempted-recon	1639 (8%)
SCAN Squid Proxy attempt	Attempted-recon	1412 (7%)
SCAN Proxy (8080) attempt	Attempted-recon	940 (5%)
WEB-IIS scripts access	Web-application activity	775 (4%)

Table 7.2: Most 5 frequent alerts

ICMP redirect host attack constituted about 30% of the total number of alerts (i.e 19451. The other most frequently occurred alert was SNMP request udp (8%) which is classified under attempted recon. Vulnerabilities in the SNMPv1 request handling of a large number of SNMP implementations allow remote attackers to cause a denial of service or gain privileges via (1) GetRequest, (2) GetNextRequest, and (3) SetRequest messages. This indicates that the attackers tried flooding the hosts and servers with SNMP requests which could made them unavailable to response to other requests prompted by various other hosts and servers on the network. It also appears that the attackers tried to gain access to the hosts and servers by sending SNMP requests to various clients on the network. This would have given them full privileges, if the request is accepted by the particular client, over that particular host or server.

On profiling the network traffic based on each individual protocol, following results were conclusive:

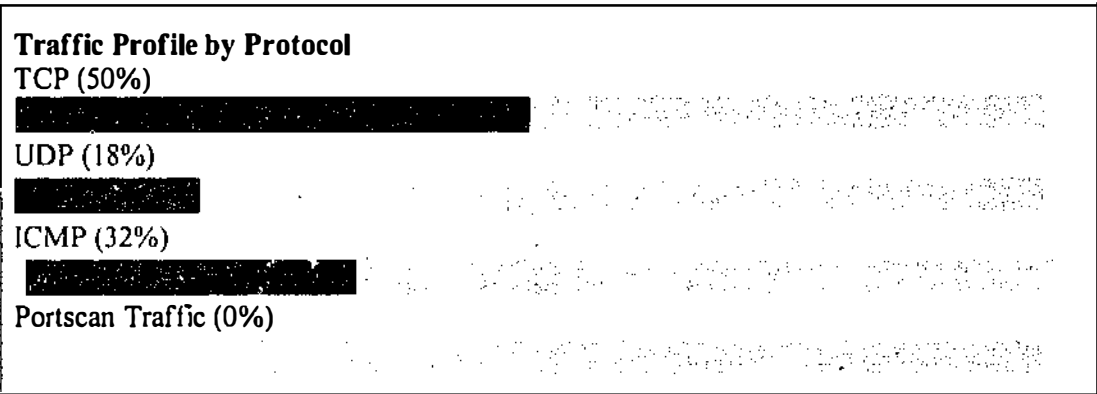


Figure 7.2: Traffic Profile by Protocol using ACID

As illustrated in figure 7.1 that majority of the network traffic was targeted to TCP (50%) and then ICMP (32%) and UDP (18%). There were also large numbers of port scans on various hosts. These port numbers can be further detailed on the basis of the occurrences of total alerts and unique alerts on them.

Port Type	Occurrences	Unique Alerts
80 /tcp	5421	195
161 /tcp	2465	4
3128 /tcp	1412	1
8080 /tcp	943	3
1080 /tcp	519	1
162 /tcp	517	3
1 /tcp	345	3
0 /udp	295	5
22 /tcp	256	10
/udp	112	1
7001 /udp	93	1
10080 /udp	81	1
10081 /udp	73	1
31337 /udp	68	1
1014 /tcp	67	7

Table 7.3: List of port numbers with occurrence of alerts

From the above table it appears that there were 3 ports which reported the maximum number of unique alerts. Those were:

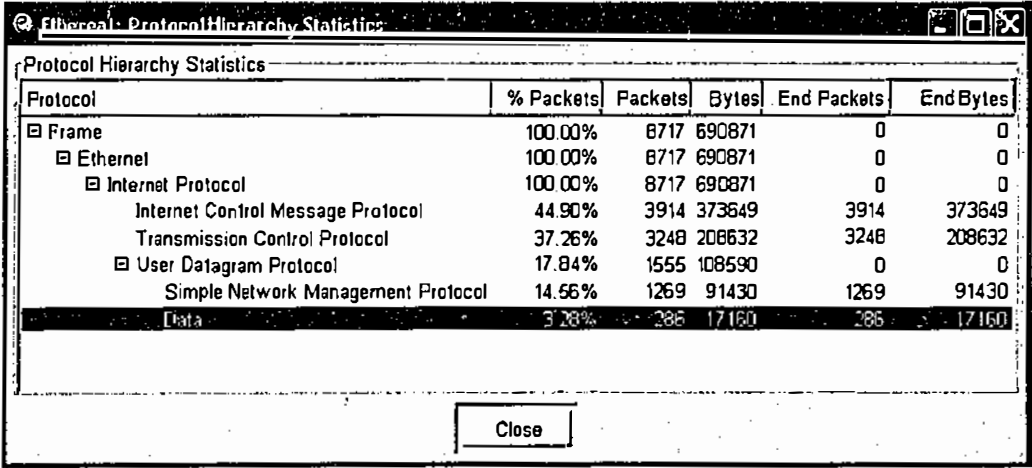
- **80 /tcp:** It is an http port which reported the maximum number of alerts (5421) as well 195 unique alerts. These unique alerts were mainly attack attempt of

exploiting WEB-MISC, WEB-CGI and WEB-IIS. These alerts occurred on 26 different hosts connected on the network.

- **22 /tcp:** This is an SSH port used for secure and encrypted transmission of data. This port reported the second most unique alerts i.e. 10. Those alerts were mainly port scans and Nmap scans. These attack attempts were targeted on 191 hosts on the network running port 22. It appears that the attacker tried to exploit this port for gathering any information travelling between the hosts.
- **1014 /tcp:** This is a kernel port which is used to implement Ethernet. There were 7 unique alerts reported on this port which were destined to 67 hosts on the network. These alerts were mainly port scans.

7.3 Ethereal Analysis

A tcpdump log file *tcpdump.log.1045547976* was generated during the first testing phase, which was analysed using Ethereal Packet sniffer. Figure 7.2 shows the Protocol Hierarchy statistic generated by Ethereal



The screenshot shows a window titled "Ethereal - Protocol Hierarchy Statistics". Inside, there is a table with the following data:

Protocol	% Packets	Packets	Bytes	End Packets	End Bytes
Frame	100.00%	8717	690871	0	0
Ethernet	100.00%	8717	690871	0	0
Internet Protocol	100.00%	8717	690871	0	0
Internet Control Message Protocol	44.90%	3914	373649	3914	373649
Transmission Control Protocol	37.26%	3248	208632	3248	208632
User Datagram Protocol	17.84%	1555	108590	0	0
Simple Network Management Protocol	14.56%	1269	91430	1269	91430
Data	3.28%	286	17160	286	17160

At the bottom of the window is a "Close" button.

Figure 7.3: Protocol Hierarchy Statistic of tcpdump.log.1045547976

As shown in the above figure 7.2, there were 8717 packets reported in the tcpdump log file out of which nearly 45% were ICMP packets and 37.26 % TCP packets. On

further analysing the log file, it was found that there were some frequent occurrences of packets over some of the ports. Those results are listed below:

- Frequent occurrences of packets of TCP SYN from port 1060 to 8080 **webcache**
- Packets with ACK, FIN, PSH and URG flags to **tcpmux**
- ACK packets to port 22 of **ssh**

On co-relating the data using ACID on port 22 of ssh it was found that there were total 256 alerts on this port out of which there were 10 unique alerts which can be further broken down as below:

4 Alerts	:- Portscans
1 Alert	:- NMAP Fingerprint (stateful) detection
1 Alert	:- Stealth Activity (FIN Scan)
1 Alert	:- Stealth Activity (NULL Scan)
1 Alert	:- Stealth Activity (SYN FIN)
1 Alert	:- Stealth Activity (Vecna Scan)
1 Alert	:-SCAN nmap TCP. This indicates that a remote user has used NMAP port scanning tool to probe the server. An NMAP TCP Ping was sent to determine if the host is reachable.

- SYN packets at port 705 for **DNM**
There were total 62 alerts on port 705 from which there was only one unique alert. It is an SNMP AgentX/ tcprequest attack (CVE: CAN-2002-0012)
- SYN packets at port 162 for Solaris. It is a binding port for Solaris systems. There were total 104 alerts on this port from which only 1 unique alert. This alert was SNMP TRAP tcp which is mainly used for causing DoS attacks. (CVE: CAN-2002-0013).

- There were some error messages too.

Error: Couldn't parse message header: wrong type for that item

- There were some UDP packets at TFTP whose information was unknown.

From the above collected data it appears that the attackers tried to make TCP connections with various hosts through number of port numbers. There were attempt of accessing the webcache which, if successful, would have given information about the accessed websites or information about any other web based services to the attackers. Using this information the attacker may be able to attempt few brute force attacks on the network. There were also few attempts on exploiting *ssh* service. It appears that the attackers tried to capture the encrypted data transferred between the hosts on the network.

There were also attempt of exploiting the SNMP services. These were mainly to cause denial of service attacks on various hosts.

7.4 Nessus Analysis

From the data files recovered from Nessus security tool (used by the attacker) various outcomes were noticed. The Nessus Security Scanner was used to assess the security of 21 hosts

- **21 security holes were found**
- **0 security warning were found**
- **79 security notes were found**

From the scan of there 21 hosts it was identified that the most dangerous service on the network was *http (80/tcp)*.

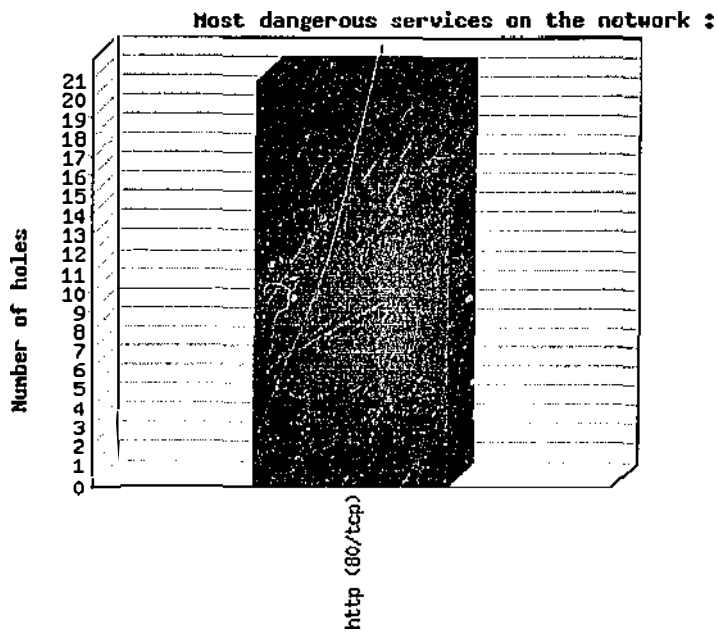


Figure 7.4: Most dangerous services on the network

Services that are most presented on the network:

There were 5 main services noticed on to the network based on the scans of 21 hosts by nessus. These services are graphically shown in the below figure on the basis of their occurrences.

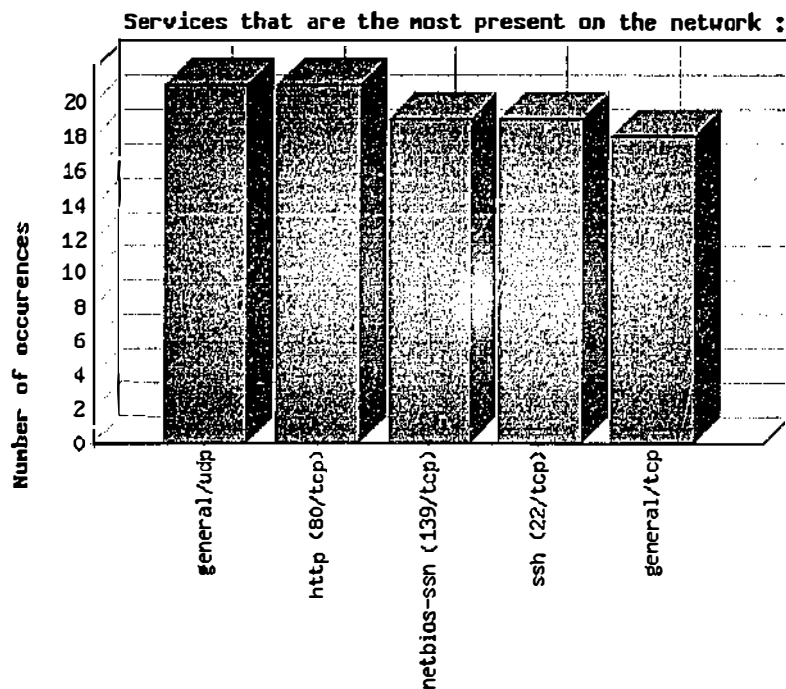


Figure 7.5: Services that are the most present on the network

On co-relating the above results with ACID following results were noticed:

general/udp -: Total number of alerts occurred on this port were 112 out of which there was only one unique alert, which was *Datum length > packet length*. This means that the data length transferred with the packet was more than the capacity of the packet. This could result in flooding the servers with the large amount of data to cause DoS attacks.

http (80/tcp) -: This port reported a very high number of alerts i.e. 5421 from which 195 were unique alerts. These unique alerts were mainly WEB-CGI and WEB IIS based probes on to the network.

Netbios-ssn (139/tcp) -: There were only 7 alerts reported on this port from which 4 were reportedly unique alerts. These alerts were merely portscans only.

ssh (22/tcp) -: SSH reported a reasonable number of alerts, which were only 256 out of which only 10 alerts were unique. These alerts comprised of various types of port scans and Nmap scans.

general/tcp -: This port just reported few Nmap based scans.

Most dangerous host on to the network:

Most dangerous host weight in the global insecurity

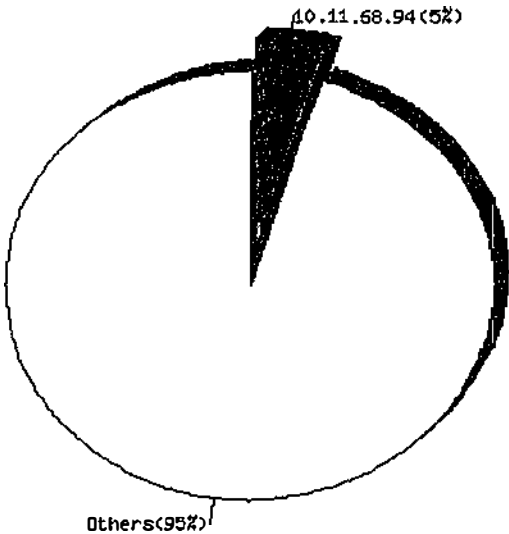


Figure 7.6: Most dangerous host on to the network

The above pie chart shows that host 10.11.68.94 was being reported the most dangerous host among the whole network.

There were also various hosts who reported a common security hole to the attacker. These hosts IP addresses are listed in the table 7.4 below:

10.11.68.94	10.11.68.136	10.11.69.81	10.11.68.103	10.11.68.157
10.11.69.174	10.11.68.92	10.11.68.59	10.11.68.117	10.11.68.238
10.11.68.91	10.11.69.83	10.11.68.116	10.11.68.192	10.11.68.90
10.11.68.253	10.11.68.19	10.11.69.117	10.11.68.95	10.11.68.89
10.11.68.18				

Table 7.4: List of Hosts with common security hole on Port 80

The above mentioned hosts reported one common security hole. The vulnerability reported by Nessus to the attacker was reported for http port 80:

The dll '/_vti_bin/_vti_aut/dwwssr.dll' seems to be present.

This dll contains a bug which allows anyone with authoring web permissions on this system to alter the files of other users.

In addition to this, this file is subject to a buffer overflow which allows anyone to execute arbitrary commands on the server and/or disable it

Solution : delete /_vti_bin/_vti_aut/dwwssr.dll

Risk factor : High

See also : <http://www.wiretrip.net/rfp/p/doc.asp?id=45&iface=1>

CVE : CVE-2000-0260

7.5 Feedback from attackers

At the end of the network penetration exercise, participants were requested to voluntarily provide their feedback about their experiences while probing the network. Those feedbacks are summarised below:

Hacker1 : “A basic network with majority of the hosts based on windows operating system. The network structure seemed confusing as the network traffic was redirected using a gateway, so there was no direct access to the routers.”

Hacker2 : “A big network with large number of hosts but majority of the hosts (which I scanned) were Windows 98 based. Seems like the organisation is lacking behind the operating system upgrades as latest windows operating systems are available which are more secure and reliable than Windows 98. No access of FTP and Telnet.”

Hacker 3 : “Basic network architecture with Windows 98 as the primary operating system on various hosts. On the AIX server, port 25 was prone to SPAM. Could not find much detail about various other services on other hosts as hardly any open ports

Windows 98 hosts were only running *http* service while few other hosts had *ssh* and *netbios*.”

7.6 Implications

The initial network design was tested successfully by various attackers. There were various types of attack attempts tried by these attackers on the hosts and servers of the network. Web-based attacks were very prominent than any other type of attack. There was also evidence of large number of port scans carried out on various hosts. After the successful completion of the penetration exercise few of the attackers voluntarily provided the feedback about their personal experience while doing the exercise. According to these attackers, the network architecture appeared too simple as it lacked any server architecture and they also suggested few changes to improve the network architecture.

On the basis of above test results there were various implications identified which needed to be changed or improved in order to enhance the deception. These implications were based on the analysis of the above results and also on the general feedback received from the attackers. These implications were:

- From the feedback report received from the attackers, it was identified that on host 10.11.69.2, AIX 3.2, port 25 was not blocked. It was advised in the report that this could be serious security concern as port 25 is mainly used for SPAM.
- The network architecture lacked the server based architecture as the network was mainly based on Windows 2000 Professional and Windows 98.
- There was no proper access provided to any host or network device. The network architecture lacked access to services like FTP or Telnet.

Changes suggested in the feedback of the attackers as well various issues identified after analysing the test data would assist in improving the created deception and would create a much better network architecture for the attackers for the purpose of second testing. Since this research is empirical in nature, the data collected after each phase of the testing will assist in improving the network for next phase of testing and analysing data.

7.7 Conclusion

On the basis of the analysis of the data obtained during the first testing and also from the feedback received from the attackers, it was concluded the initial network required various changes.

The level of deception appeared to be very low which raised suspicion among the attackers. It was a very low level interaction network which provided very few opportunities to the attackers. Also there were no remote login services available on the network, which are typically expected on any corporate network.

There were issues which needed to be addressed for implementing a successful and improved deception for the attackers. On the basis of the feedback of attackers received and also according to the analysis of the results, it was found that the Cisco routers did not have any remotely based console access. Also it was required to provide a server type network architecture to the network. Therefore, Windows 2000 operating system was replaced with Windows NT operating system. Remote access services like FTP and Telnet needed to be implemented in order to present a improved deception to the attackers on the network by offering them more support for attack.

8. Second Test results on Honeyd 0.5

8.1 Overview from first test results

After the first testing phase, there were few security holes and alerts that potentially were identified. Also on the basis of the general feedback received from the hackers regarding the network architecture, it was found that it was a very low interaction network which did not give many opportunities for interaction to attackers to interact with. From the hackers' point of view, it was a secure network with limited level of interaction. Therefore, the network configuration was improved based on the previous data analysis. Following changes were made before the second test phase began:

- Honeyd0.4a was upgraded to Honeyd-0.5. This was mainly done because the latest version of honeyd had some extra features such as separate logging facility and also is capable of using fingerprint signatures of Xprobe fingerprinting tool.
- Arpd0.1 was upgraded to Arpd0.2
- Honeyd configuration file (Appendix C.2) was amended. Following changes were made in the configuration file:
 - Change in IP addresses from 10.x.x.x to 192.168.x.x. This was mainly done because the existing 10.x.x.x network configuration was not compatible with the new improved version of the Honeyd-0.5.
 - Host with “Windows 2000 Professional, Build 2128” was replaced by “Windows NT 4.0 Server SP5-SP6”. This was mainly done to provide a server type of architecture to the network, which was previously missing. Additionally, a perl script that emulated IIS server was implemented on http port 80. There was also various ports left open on this particular host, such as, tcp port 137, 139 and udp ports 137, 135.

These ports are used by Net-bios on the servers for network connections for input-output services.

- Over the “AIX 3.2” host, port 25 was blocked. This is a standard practice by various organisations and ISPs to block port 25 as this is used to send emails. This helps in cutting down the SPAM emails. This too enhanced the network architecture which resembles to a corporate network. Also port 21 for FTP was opened with a shell script (ftp.sh, see Appendix C.6) running on this port. This script provided the anonymous log in on the ftp server with restricted access as ‘guest’. FTP is an attractive service for the attackers as using ftp would allow them to upload their programs and tools from remote locations on to the network.
- Both a Cisco router and a switch were provided with telnet connections. On telnet port 23, a “router-telnet.pl” script (see Appendix C.8) was implemented which provided a telnet access to the router. This would provide a console based access to the router using telnet for the attackers.
- “Novell Netware 3.12 or 386 TCP/IP” was replaced with the latest signature of “Novell Netware 5.0 SP5”.

The above mentioned changes provided more realistic appearance to the network and assisted in eliminating any type of suspicion, regarding the presence of deception, which could arise in the minds of attackers while probing and attacking the network.

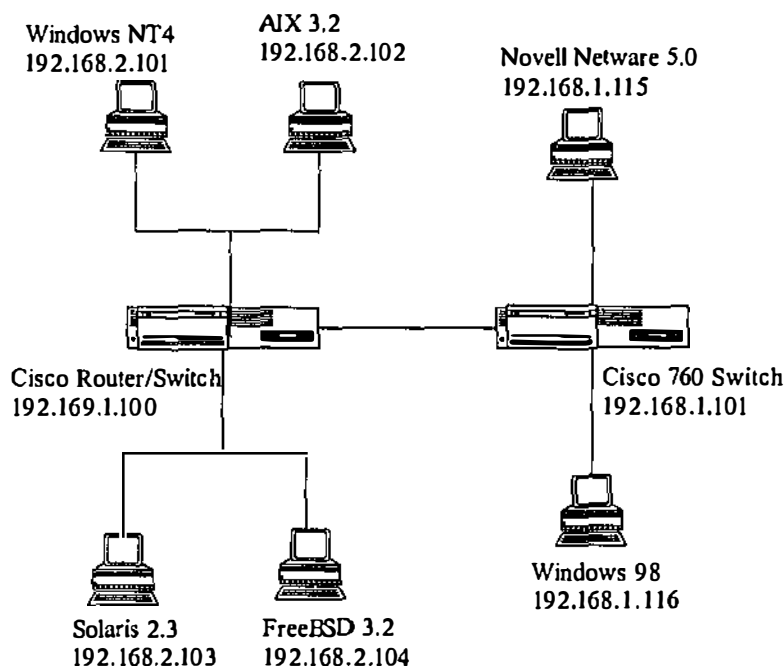


Figure 8.1: Improved Honeynet Architecture

The table 8.1 below summarises the allocated IP addresses and services on the new improved Honeyd virtual network:

IP Addresses	Signatures	Services
192.168.2.101	Windows NT 4.0 Server SP5-SP6	TCP – port 80, 137, 139 UDP – port 135, 137
192.168.2.102	AIX 3.2	TCP – port 21, 80
192.168.2.103	Solaris 2.3 – 2.4	TCP – port 80
192.168.2.104	FreeBSD 3.2 – 4.0	TCP – port 80
192.168.1.100	Cisco IOS 11.3 – 12.0(11)	TCP – port 23
192.168.1.101	Cisco Router/Switch with IOS 11.2	TCP – port 23
192.168.1.115	Novell Netware 5.0 SP5	TCP – port 80
192.168.1.116	Windows 98	TCP – port 80
Default	Windows 98	TCP – port 80, 22, 139

Table 8.1: Summary of IP addresses allocated on corporate network

The telnet service was only available on the Cisco routers (192.168.1.100-101). This was due to the fact that Cisco routers are accessible via console through telnet. This may provide an opportunity for the attackers to exploit the telnet service to gain access to the routers. The *Default* Windows 98 systems had port 22 open for *ssh*, to

communicate between other hosts and servers in a secure manner. The reason researcher decided to have port 22 open on the network was mainly due to the fact that the attackers may try to sniff the encrypted data (such as passwords) communicating between the various hosts and servers of the network.

8.2 Second Test Results

During the second testing phase, a group of students (see section 3.4) were asked to probe the improved network designed (192.168.1.0/24 and 192.168.2.0/24, Figure 8.1) using the honcyd 0.5. Similar to previous testing, the attackers were asked to take their places in one single laboratory and also they had their own laptops with their own choice of programs and tools required for the penetration exercise. This testing phase existed between the time window of 21-03-2003 [00:54:29] to 22-03-2003 [12:21:55]. There were various mechanisms tried by students which generated a substantial amount of data in the log files. The logs generated during this period were also stored on a database which was used by ACID (Analysis Console for Intrusion Databases) for the analysis purpose. There were also few tcpdump logs which were analysed using Ethereal. The honeyd log files were analysed using Analyst, but the charts generated by Analyst were too big to fit on paper therefore they have been included on the provided CD.

ACID Analysis

From the initial testing the Snort IDS recorded 23500 alerts, out of which there were 554 unique alerts which were divided into 14 different categories.

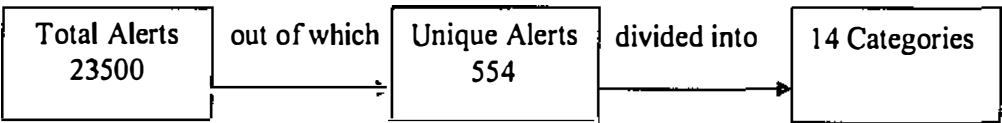


Figure 8.2 Acid Analysis

These 14 categories are given below:

Classification	Total Alerts	Signatures
Unclassified	1486 (6%)	266
Bad-unknown	9466(40%)	5
Attempted-recon	5615 (24%)	89
Web-application activity	3914 (17%)	112
Web-application attack	2641 (11%)	60
Misc-activity	125 (1%)	2
Misc-attack	14 (0%)	1
Attempted-dos	106 (0%)	6
Protocol-command-decode	23 (0%)	1
Attempted users	21 (0%)	2
Successful admin	44(0%)	1
Attempted-admin	20 (0%)	3
Unknown	5 (0%)	1
Rpc-portmap-decode	20 (0%)	4

Table 8.2: 14 Alert Categories

The above classified alerts mentioned in table are described in detail below:

Unclassified

There were 266 signatures recorded under this classification. These signatures were different port scans on various hosts and servers of the network. As described earlier in the chapter 7, these scans can assist attackers in exploiting and probing the most vulnerable ports and services which may provide them with access to the whole network.

Bad-unknown

There were 5 signatures which were identified under this category. They were:

- *ICMP Redirect host*

This comprised of 36% of the total number of alerts recorded over the network. This may have occurred because all the network traffic was routed through a gateway machine within the laboratory which further forwarded the network traffic to their respective destinations (explained in chapter 7).

- *Attack Responses http dir listing*

It is a type of snort alert which indicates that a web server is responding to the probes with a reply. From the collected data, it is found that there were 6 source IP addresses which responded back to the single attacking IP (172.16.255.253). Therefore, it seems that these hosts were responding back to the attacker's machine about them being active on the network.

- *Attack Responses id check returned root*

It is also a type of snort alert which is triggered when someone use a root as it check the payload for word "root" from any network and that packet has the ACK flag set. There was just a single attempt of this attack from one source address to one destination address.

- *MISC Tiny Fragments*

There were few tiny fragments of packets transmitted from one source address to 3 different destination addresses. This indicates that one of the attackers tried to carry out a DoS attack by sending small fragments of packets to different hosts. This could have made these hosts unavailable, if the attack was successful, for any further network requests.

- *MISC Large UDP packets*

Large UDP packets were used by the attackers to flood some of the hosts with UDP request to attempt a DoS type of attack on the particular hosts and servers. This could be mainly to bring down the various servers and hosts on the network which eventually would have collapsed the whole network.

Attempted-recon

There were 89 different signatures which were identified in this category. Some of the most prominent were (discussed in detail in chapter 7):

- *SNMP request UDP (CAN 2002-0012)*

As explained in chapter 7, a large number of SNMP implementations allow remote attackers to cause denial of service or gain privileges via SNMPv1 trap handling.

It was identified that about 17 hosts were subject to this attack by one single attacker. It appears that the attacker made an attempt of implementing a DoS attack on those hosts which would have made them unavailable for other network requests.

- *Large number of WEB-CGI and WEB-MISC alerts.*

- *SCAN Socks Proxy attempt*

As described in chapter 7, SOCKS protocol provides a support for developing secure communication using different security technologies. It acts like a medium when a client tries to connect to any application server.

It was identified that there were SCAN SOCKS Proxy attempt on 17 destination hosts. It appears that the attackers tried to probe the SOCKS proxy to connect to various servers available on the network.

Web-application activity and Web-application attacks

There were 112 and 60 unique signatures in both categories respectively. These signatures were focused on either IIS Script access based attacks or CGI and MISC web based attacks. On the basis of the data collected, it appears that web based attacks were the most common form of attacks used by attackers. More than 180 signatures were identified which were used or attacking or probing the web based services and port numbers.

Misc-activity

There were only 2 signatures reported under this category:

- *Experimental MISC AFS access*

There were miscellaneous attempts of accessing AFS by attackers using Nessus scanning tool. This type of attempt, if successful, can provide full access to the filesystem of the hosts and can cause any damage to the system. According to ACID, this attack was carried on 17 different destination hosts from one single attacking IP.

- *BAD traffic tcp port 0 traffic (CVE 1999-0675)*

This attack attempt is associated with Checkpoint Firewall-1. It is an attempt to carry out a Denial of Service (DoS) attack on the firewall via UDP packets. This attack attempt was implemented using Nessus scanning tool. This attack was just attempted on 2 destination addresses in compare to 135 hosts in previous testing.

Misc-attack

There was a single Experimental Web-PHP directory.php access attempt (CAN 2002-0434) under this category. It allows remote attackers to execute arbitrary commands via shell meta characters in the *dir* parameter. This attack was destined to 13 different hosts on the network. This is also a type of attack which Nessus tries to execute on the systems connected to the network.

Attempted-dos

There were a total of 6 signatures reported under this category from which 5 signatures were identified as DDoS attack signatures (explained in chapter 7). It was identified that this distributed denial of service attack was attempted on 18 different hosts on the network by 2 different source addresses. The remaining 6th signature was:

- *DOS Bay/Nortel Nautica Marlin (CVE: 2000-0221)*

The Nautica Marlin Bridge allows remote attackers to cause a DoS via a zero length UDP packet to the SNMP zero. A DoS attack is attempted by transmitting UDP packets to the systems.

Protocol command decode

There was a single Experimental MISC IPsec PGPNet connection attempt destined to 18 different hosts on the network. This attack is associated with Pretty Good Privacy (PGP) program that provides electronic mails with Privacy by encrypting the contents of the e-mails. From the above attack it appears that the attacker tried to connect to the PGPNet to sniff the encrypted emails which may have provided him or her with some information. This is one of the probes which Nessus scanning tool explores while scanning the network in order to exploit the PGPNet, if it exist on the network.

Attempted users

There was 2 attack signatures reported under this category. They were:

- *WEB-CGI webspd access (CVE: 2000-0127)*

The webspd configuration program does not properly disable access to the WSMAdmin (WebSpeed Messenger Administration tool) utility, which allow remote attackers to gain privileges. It was identified that this attack attempt was destined to 15 different hosts on the network.

- *RSSERVICES rsh bin*

This event indicates that an attempt was made to login to an rshd server (remote shell) using the *bin* account. This event occurred from 2 different attacking IP addresses towards a single host IP 192.168.1.1

Successful admin

This category included only a single alert related to TFTP GET passwd. It uses UDP which provides no security features (explained in chapter 7). It was reported from 2 source addresses to 17 other destination hosts on the network. It appears that the attackers tried to retrieve the system passwords by connecting to TFTP using UDP packets with GET command.

Attempted admin

There were 3 alert signatures reported under this category. These were:

- *RSERVICES* rsh root
This event indicates that an attempt was made to login to an rshd server (remote shell) using the *root* account.
- *RSERVICES* rlogin root
This event is similar to the previous event which if successful, could indicate a root compromise.
- *RSERVICES* rsh froot
This event may provide root access to any AIX 3.2.x system. Since there was one AIX 3.2 server in the network architecture, it appears that the attackers tried to probe the AIX system to gain root access on it.

Unknown

There was only one unknown signature based on X11 xopen. This event indicates that an external user has attempted to launch an X11 application on an internal X server.

RPC portmap-decode

There was 4 signatures reported each related to different type of RPC requests, which are:

- *RPC portmap listing*
This event indicates that a query was sent to the rpcbind/portmap daemon on a solaris machine, requesting port information for rpc-services.
- *RPC portmap request yppasswdd*
The rpc.yppasswdd server is used to handle password changes request from yppasswdd and modify the NIS (Network Information Service) password file. A Buffer flow vulnerability exists in the rpc.yppasswdd utility. The problem occurs due to insufficient bounds checking before copying remotely-supplied

user information into a static memory buffer. As a result, a malicious user may be capable of exploiting this issue to overwrite sensitive locations in memory and thus execute arbitrary code with super user privileges.

- *RPC portmap request rstatd*

A query was sent to the portmap daemon, requesting port information for the rstatd service. The rstatd daemon can give detailed information about the host.

- *RPC portmap request mountd*

This event indicates that a query was sent to the portmap daemon, requesting port information for the rpc.mountd service. This query usually precedes attempts to access mountd, access NFS, or to attack the rpc.mountd service with protocol or buffer overflow conditions.

Conclusion

From the above discussions, it appears that the attackers tried various attacking methods to attack the network. Similar to previous testing, there were high amount of portscans as well as lot web based attacks using CGI scripts and IIS access attempts. There were also few occasions when remote login attempts were also noticed. There was evidence which illustrates that the attackers tried gaining *root* access on some of the hosts on the network. There was an event which indicated that a query was sent to the rpcbind/portmap daemon on a solaris machine, requesting port information for rpc services using RPC portmap listing. Most of the attacks were very similar to those that were attempted in previous testing which indicates that the most of the attacks were consistent on the network. Therefore, from the above discussion the most frequent 5 alerts were:

Port Type	Occurrences	Unique Alerts
80 /tcp	8633	255
161 /tcp	2950	5
/tcp	960	1
162 /tcp	250	3
10080 /udp	114	1
177/udp	114	1
7001 /udp	114	1
10081 /udp	114	1
31337 /udp	96	1
22 /tcp	84	26
1 /tcp	62	4
69/tcp	47	2
800/udp	39	1
3128/tcp	33	1
8080/tcp	24	1

Table 8.4: List of popular destination port numbers with occurrence of alerts

From the above table 8.4 it appears that there were 2 ports which reported the maximum number of unique alerts. Those were:

- **80 /tcp:** It is an http port which reported the maximum number of alerts (8633) as well as 255 unique alerts. These unique alerts were mainly attack attempt of exploiting WEB-MISC, WEB-CGI and WEB-IIS. These alerts occurred on 24 different hosts connected on the network from various attacking IP
- **22 /tcp:** This is an SSH port used for secure and encrypted transmission of data. This port reported the second most unique alerts i.e. 26. Those alerts were mainly port scans and Nmap scans. These attack attempts were targeted on 36 hosts on the network running port 22. It appears that the attacker tried to exploit this port for gathering any information travelling between the hosts.

8.3 Findings from log files

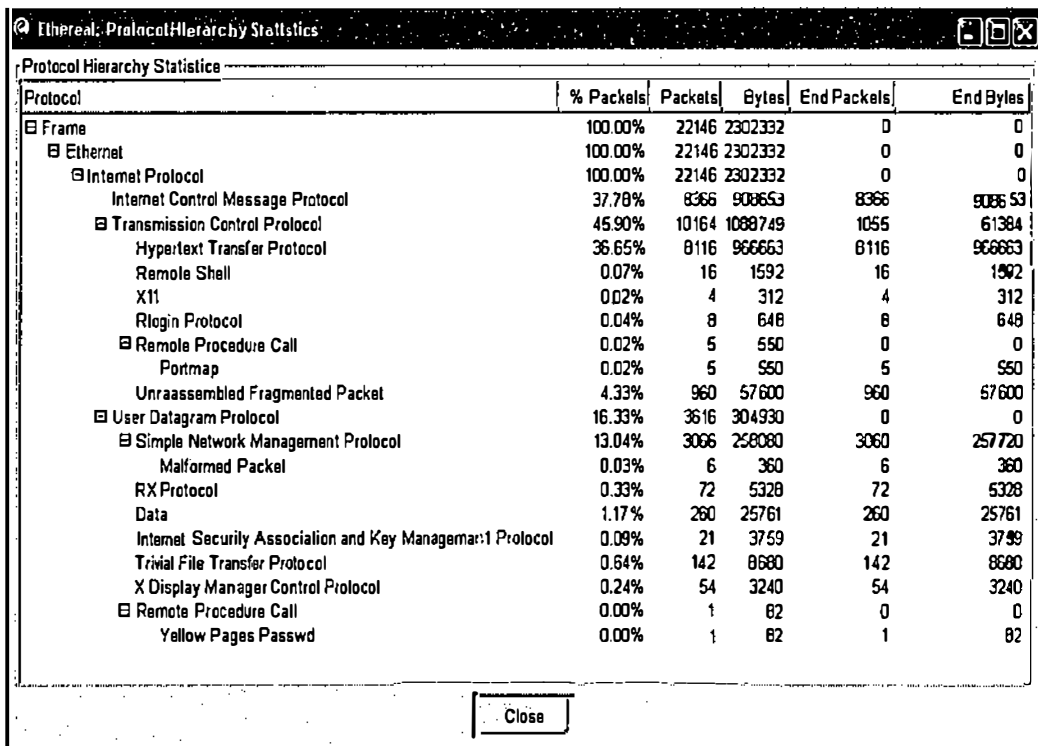
From the study of log files, it was identified that most of the attackers used Nessus (<http://www.nessus.org>) as the network scanning tool to identify the vulnerabilities in the network. On the findings of the vulnerabilities most of them tried to conduct brute

force attack over the network. There were various probes for exploiting the vulnerabilities in web based services and ports. Various CGI and perl scripts were attempted in order to gain entry into web based services.

There were various attempts of exploiting vulnerabilities of SSH using Nessus and Putty-Release-0.53b. PuTTY is a free SSH, Telnet and Rlogin client for 32-bit Windows systems

8.4 Ethereal Analysis

There were two tcpdump log files generated during the second testing phase (*tcpdump.log.1048208266 [created on 21/03/03]* and *tcpdump.log.1048298249 [created on 22/03/03]*), which was analysed using Ethereal Packet sniffer. Below are the Protocol Hierarchy statistics generated by the Ethereal:



Protocol	% Packets	Packets	Bytes	End Packets	End Bytes
Frame	100.00%	22146	2302332	0	0
Ethernet	100.00%	22146	2302332	0	0
Internet Protocol	100.00%	22146	2302332	0	0
Internet Control Message Protocol	37.78%	8366	908653	8366	908653
Transmission Control Protocol	45.90%	10164	1088749	1055	61384
Hypertext Transfer Protocol	36.65%	8116	966663	8116	966663
Remote Shell	0.07%	16	1592	16	1592
X11	0.02%	4	312	4	312
Rlogin Protocol	0.04%	8	648	8	648
Remote Procedure Call	0.02%	5	550	0	0
Portmap	0.02%	5	550	5	550
Unassembled Fragmented Packet	4.33%	960	57600	960	57600
User Datagram Protocol	16.33%	3616	304930	0	0
Simple Network Management Protocol	13.04%	3066	258080	3060	257720
Malformed Packet	0.03%	6	360	6	360
RX Protocol	0.33%	72	5328	72	5328
Data	1.17%	260	25761	260	25761
Internet Security Association and Key Management Protocol	0.09%	21	3759	21	3759
Trivial File Transfer Protocol	0.64%	142	8680	142	8680
X Display Manager Control Protocol	0.24%	54	3240	54	3240
Remote Procedure Call	0.00%	1	82	0	0
Yellow Pages Password	0.00%	1	82	1	82

Figure 8.4: Protocol Hierarchy Statistics of *tcpdump.log.1048208266*

As shown in the above figure, there were 22146 packets reported in the tcpdump log file *tcpdump.log.1048208266* out of which nearly 37.78% were ICMP packets and 45.90 % TCP packets. Majority of the TCP packets were http packets i.e. 36.65% of the total TCP packets. There was also evidence of Remote Shell and Rlogin protocol packets. About 16.33% of total packets consist of UDP packets from which 13.84% were SNMP packets. On further analysing the log file, it was found that there were some frequent occurrences of packets over some of the ports. Those results are listed below:

- There was an attempt of remote login as *root* from 172.16.253.253 (attacker's or source IP address) to 192.168.1.1 (destination IP address)
- There was queries related to TFTP (Trivial File Transfer Protocol) from IP address 172.16.253.253 to 192.168.1.1. There was also a read request from TFTP for file */etc/passwd*.
- Frequent occurrences of packets of TCP FIN, PSH and URG from port 42778 of 172.16.253.253 to various ports of 192.168.1.1

Ethereal: Protocol Hierarchy Statistics					
Protocol Hierarchy Statistics					
Protocol	% Packets	Packets	Bytes	End Packets	End Bytes
[-] Frame	100.00%	746	363340	0	0
[-] Ethernet	100.00%	746	363340	0	0
[-] Internet Protocol	100.00%	746	363340	0	0
Internet Control Message Protocol	14.61%	109	34649	109	34649
[-] Transmission Control Protocol	76.94%	574	80973	38	2756
[-] Remote Procedure Call	1.34%	10	1100	0	0
Portmap	1.21%	9	990	9	990
RSTAT	0.13%	1	110	1	110
Remote Shell	0.40%	3	244	3	244
X11	0.13%	1	78	1	78
Rlogin Protocol	0.27%	2	162	2	162
Hypertext Transfer Protocol	69.71%	520	76633	520	76633
[-] User Datagram Protocol	8.45%	63	247718	0	0
[-] Simple Network Management Protocol	1.07%	8	480	0	0
Malformed Packet	1.07%	8	480	8	480
Data	6.03%	45	246264	45	246264
[-] Remote Procedure Call	0.54%	4	360	0	0
RSTAT	0.13%	1	82	1	82
YellowPages Passwd	0.13%	1	82	1	82
Portmap	0.27%	2	196	2	196
Internet Security Association and Key Management Protocol	0.27%	2	396	2	396
Trivial File Transfer Protocol	0.54%	4	256	4	256

Figure 8.5: Protocol Hierarchy Statistics of tcpdump.log.1048298249

As shown in the above figure 8.4, there were 746 packets reported in the tcpdump log file *tcpdump.log.1048298249* out of which nearly 14.61% were ICMP packets and 76.94% TCP packets. Majority of the TCP packets were *http* packets i.e. 69.71% of the total TCP packets. There was also evidence of Remote Shell and Rlogin protocol packets in this log file too. About 8.45% of total packets consist of UDP packets. On further analysing the log file, it was found that there were some frequent occurrences of packets over some of the ports. Those results are listed below:

- There was evidence of using program Portmap Version 2 with procedure DUMP from source IP address 172.16.1.120 towards destination IP address 192.168.1.1. These packets were sent from source port 624 towards destination port sunrpc (port 111).
- There were few UDP packets from 172.16.0.1 source port *nfsd* towards 172.16.1.120 port 800 with large amount of data (around 4216 bytes).

- There were also few packets of Yellow Pages Password (YPPASSWD) of program version 32803 using Remote Procedure Call (RPC) version 2
- There was an attempt of remote login as *root* from 172.16.1.120 (attacker's or source IP address) to 192.168.1.1 (destination IP address)
- There were queries related to TFTP (Trivial File Transfer Protocol) from IP address 172.16.1.120 to 192.168.1.1. There was also a read request from TFTP for file */etc/passwd*.

From the above results it appears that the attackers repeatedly tried to remotely login on to the network to gain root access on the host machines. There were multiple attempts for gaining passwords from the network as the attackers tried using Yellow pages password programs and also tried TFTP which is not connection oriented and uses UDP packets for sending packets. The attackers tried connecting to the password files on the host machines to retrieve them. It appeared that host 192.168.1.1 was the prime target for the attackers as there were multiple attacks noticed on this particular host. There were also evidence of one of the attackers connecting to 192.168.1.1 host on to the network but could not gain much access on it. This particular attacker tried to exploit Network File System (NFS) of the host to connect to the host.

8.5 Analyst Notebook 6 graphs

The log files generated by honeyd were imported into the Analyst Notebook 6 to generate the graph of relationship between various hosts on the network. The graph displays various source hosts connecting to destination hosts with linking them according to their occurrences on the network. Since the generated graphs were large in size therefore, they are included on the provided CD along with the chart viewer program. See appendix E for chart file details

8.6 Feedback from Attackers

Similar to the previous penetration exercise, participants were asked to voluntarily provide their personal feedback about the network and their findings. Below are the feedback received from some of the attackers:

Hacker1: “Created a footprint of the network and located few prime targets using traceroute and pings. It appeared to be a very big network with large number of hosts and few servers. There were lot of MAC address request on the network. From the results of traceroute, it appeared that host 192.168.1.1 was connecting to the subnet. Therefore, I targeted my attacks mainly on host 192.168.1.1 as it appeared to be in good position to be a router. It was also sharing some NFS paths on to the subnet. After some deep probing on host 192.168.1.1, the situation became a bit confusing. It appeared that there were 2 hosts on same IP address 192.168.1.1 which put me on a backfoot while probing the network. I also discovered a Cisco router on the network which was vulnerable to DOS attacks.”

Hacker2 -: “Large number of hosts with Windows 98 operating system which were running SSH servers. It appeared that a vulnerable SSH versions were running on the network. The entry point to the network was through the Cisco Router running on 192.168.1.100. But when conducted few traceroutes on various other hosts, it appeared to trace over 192.168.1.1. It appeared that 192.168.1.1 was acting as a gateway machine which was redirecting the network traffic to the subnets. Overall it appeared to be a large network with limited vulnerabilities to exploit.”

8.7 Implications

After the improvisation of the network, it presented various services to the attackers. The data collected from the testing was of much higher value than the previous testing as evidences of some new types of attack attempts made by the attackers were recovered. Most of the attack attempts were similar to what was attempted in the previous testing. There were again a high number of attempts on web-based services and ports such as http port 80. It appears that the *http port 80* is one of the most

prominent targets for the attackers. Also in the previous testing, Nessus reported that http port 80 was the most dangerous service on the network therefore there was slightly an increase of attack attempts on port 80 in the second tests. There were also evidences of attempting remote login on the improved network which was hardly attempted in the previous tests. The second tests were more focused on selected hosts rather than the whole of network as was the case in the initial tests. During the second tests, host 192.168.1.1 was one of the prime targets for the attackers as there was various attack attempts noticed and alerted on this host.

Also during the second tests there were evidences of using third party programs such as PuTTY and Portmap. PuTTY was mainly used in exploiting the SSH service at port 22 and Portmap was used for mapping solaris system for requesting port information. Use of third party programs was not noticed during the initial testing of the network.

Although the new network configuration provided a better network architecture and higher level of deception as the attackers were successfully disguised with the emulated services offered on the network, so there was limited changes required in the network for improving the level of deception. During the penetration exercise one of the attacker (who did not provided any feedback) seemed to be growing suspicious about one of the service offered on the network. It was noticed that hosts which were operating on Windows 98 were running Microsoft IIS v.5, which is not possible. Therefore it created some doubts among the attackers. Hence, the shell script which was emulating IIS v.5 was needed to be changed to emulate IIS v.4 for more realistic appearance of the hosts.

9. Third Test results on Honeyd 0.5

9.1 Overview from second test results

After the second testing phase, there were potential security holes and alerts that were identified. On the basis of the general feedback received from the hackers regarding the network architecture, it was found that the network defences were well in place and did not provided much opportunity for an attacker to do over the network. Therefore, it may not be wrong to assume that the created deception has been so far successful in deceiving the attackers as they seemed to believe it to be a legitimate organisational network. One important point noted by an attacker was that on Windows 98 he found that Microsoft IIS v.5 was running which is not possible. Therefore the only change made to the configuration file was changing the script for Windows 98 operating system for Microsoft IIS v.4. (See Appendix C.4 for script)

9.2 Third Test Results

During the third testing phase, a group of students were asked to probe the improved network designed (192.168.1.0/24 and 192.168.2.0/24) using the honeyd 0.5. This testing phase existed during the time window from 26-05-2003 [14:09:16] to 27-05-2003 [18:14:50]. There were various mechanisms tried by students which generated a substantial amount of data in the log files. The logs generated during this period were also stored on a database which was used by ACID (Analysis Console for Intrusion Databases) for the analysis purpose. There were also few tcpdump log files which were analysed using Ethereal. Syslogs and temporary log files were analysed manually while honeyd log file was analysed using Analyst Notebook 6.

From the initial testing the Snort IDS recorded 19036 alerts, out of which there were 453 unique alerts which were divided into 12 different categories.

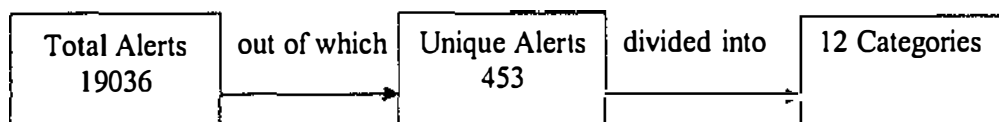


Figure 9.1 Acid Analysis

These 12 categories are given below:

Classification	Total Alerts	Signatures
Unclassified	342 (2%)	211
Bad-unknown	11737(62%)	4
Attempted-recon	4814 (25%)	81
Web-application activity	1299 (7%)	104
Web-application attack	664 (3%)	40
Misc-activity	151 (1%)	3
Misc-attack	1 (0%)	1
Attempted-dos	15 (0%)	5
Protocol-command-decode	3 (0%)	1
Successful admin	6(0%)	1
Attempted-user	1 (0%)	1
Rpc-portmap-decode	3 (0%)	1

Table 9.1: 12 Alert Categories

The above classified alerts mentioned in table are described in detail below:

Unclassified

There were 211 signatures recorded under this category. These signatures were mainly port scans on various ports of different hosts on the network. As described previously in chapter 7 and 8, attackers perform port scans to identify various exploits and fingerprint the network.

Bad-unknown

There were 4 alert signatures reported under this category. These signatures were:

- *ICMP Redirect host*

This comprised of 34% of the total number of alerts recorded over the network. Since all the network traffic was routed through a gateway machine, the total number of redirect alerts was high.

- *TFTP Get*

This event indicates that the attackers tried using TFTP to retrieve information such as passwords using the GET command. Since TFTP uses UDP based connections, it does not provide any security feature.

- *MISC Large UDP packets*

Large numbers of UDP packets were transmitted from one single host (192.168.1.28). This UDP traffic constituted about 28% of the total alerts recorded on the network. Such attack was mainly attempted for a DoS attack on that particular host. This may have resulted in the network traffic flow delay or complete halt of the network traffic to this particular host.

- *Attack Responses http dir listing*

It is a type of snort alert which indicates that web server is responding to the probes with a reply. From the collected data, it is found that there were 2 sources IP addresses which responded back to the 2 other destination IP addresses.

Attempted-recon

This category consisted of 81 different signatures from which the most prominent were:

- *SNMP request udp (CAN 2002-0012)*

Vulnerabilities in large number of SNMP implementations allow remote attackers to cause denial of service attacks or gain privileges via SNMPv1 trap handling. This attack attempt consisted of 4% of the total number of alerts and was destined to 219 different hosts on the network. There were large numbers of SNMP requests which were flooded on these hosts which could have caused DoS attack on these hosts and would have made them unavailable for other requests.

- *SCAN SOCKS Proxy attempt*

As described in previous chapters 7 and 8, SOCKS provides support for developing secure communications by integrating other security technologies. It is a medium between the client and the application servers. It was identified that there were SCAN SOCKS Proxy attempt on 129 different hosts from 3 different source IP addresses. It appears that the attackers tried to probe the SOCKS proxy to connect to various servers available on the network.

- *SCAN Squid Proxy attempt*

Squid is full-featured Web proxy cache designed to run on UNIX systems. About 6% of the total alerts were Squid proxy scans which were destined to 125 different hosts from 5 different source IP addresses. It indicates that the attackers tried to gain access to the web cache in order to gain some valuable information such as passwords stored in cache for gaining privileges over the hosts.

- *WEB-MISC and WEB-CGI alerts*

A large number of alerts were reported based on web-based services and port numbers.

Misc-activity

There were 3 signatures under this category:

- *BAD TRAFFIC bad frag bits*

The attacker tried to send large amount of UDP packets to flood the hosts and servers to attempt a denial of service attack. This indicates that the attackers tried to flood the servers and hosts to disrupt the network services and make them unavailable for any further network service requests.

- *BAD Traffic udp port 0 traffic (CVE 1999-0675)*

As mentioned in chapter 7 and 8, this alert is associated with Checkpoint Firewall-1 vulnerability. It is subjected to DoS types of attacks via UDP packets. This vulnerability is exploited using Nessus by sending large number

of UDP packets on the hosts. This alert originated from 3 different source addresses and was destined to 68 different hosts on the network.

- ***EXPERIMENTAL MISC AFS access***

As described in chapter 7 and 8, miscellaneous attempts of accessing AFS, a distributed file system, was made using Nessus. This would provide the attacker full access to the file system of the host machine and can cause any level of damage to the system. But in the 3rd testing it was not attempted extensively on the network as it was just reported to one destination address whereas in the first testing it was reported to 21 different hosts on the network.

Web-application activity and web application attacks

There were 104 and 40 unique signatures in both categories respectively. These signatures were focused on either IIS Script access based attacks or CGI and MISC web based attacks.

Attempted-dos

This category had 5 signatures associated with the attempt of DDoS attacks using mstream, shaft or Trin00. These attacks are classified under CAN 2000-0138 of CVE which states that “a system has DDOS attack master, agent or zombie installed such as 1) Trin00, 2) Tribe Flood Network (TFN), TFN2K, 4) stacheldraft 5) mstream and 6) shaft. If this attack was successful, it would have collapsed the whole network and would have made it inaccessible for any user.

Protocol-command-decode

There was only one alert reported under this category which was Experimental MISC IPSec PGPNet connection attempt.

Successful-admin

There was only one alert reported under this category which was TFTP GET passwd. As described in chapter 7 and 8, it uses UDP connections to retrieve password files from the servers or remotely boot the workstations.

Misc-attack

There was only one alert reported under this category which was Experimental WEB-PHP directory.php access (CAN 2002-0434). It allows remote attackers to execute arbitrary commands via shell meta characters in the *dir* parameter.

Attempted-user

There was only one alert reported under this category which was WEB-CGI webspeed access. This attack does not disable the WebSpeed Messenger Administration (WSMAdmin) utility properly thus allows remote attackers to gain privileges.

Conclusion

Similar to previous testings, there were high amount of port scans as well as lot web based attacks using CGI scripts and IIS access attempts. From the above table we can see that there were 104 signatures under web-application-activity and 40 signatures under web-application-attack. Most of the attacks were very similar to previous attacks conducted in last 2 network penetration exercise. There was large number of UDP packets flooded on host 192.168.1.28. This was mainly conducted to carry a DoS attack on the host and make it unavailable for other network requests. One another important point to note from the above data is that 62% of the total alerts were under Bad-unknown classification. This means that most of the attack attempts were either ICMP redirect request of Misc large UDP packets to carry out denial of service attacks. Therefore, out of the total alerts, the most frequent 5 alerts were:

Signature	Classification	Total
ICMP redirect host	Bad-unknown	6478 (34%)
MISC large UDP packets	Bad-unknown	5242 (28%)
Scan Squid Proxy Attempt	Attempted-recon	1061 (6%)
Scan Proxy (8080) attempt	Attempted-recon	730 (4%)
SNMP request udp	Attempted-recon	690 (4%)

Table 9.2: Most 5 frequent alerts

ICMP redirect host attack constituted about 34% of the total number of alerts. This type of attack is capable of crashing or locking up a host machine. The other most frequently occurred alert was 'Misc Large UDP packets' which is also classified under *bad-unknown*. This event indicates that an abnormally large UDP packet was sent to your server. This may indicate a denial of service attack or the use of a covert channel. Since this event was caused by a UDP packet, the source IP address could be easily forged. Also, it has been noted that due to the nature of this event the attacker does not normally require response traffic. In most cases this means that the event should be analysed along with other supporting data before acting on the event.

The other three alerts were classified under *attempted-recon*. On comparing the results of Table 12.1 and 12.2, it can be noted that the majority of the network traffic was classified under *bad-unknown* and *attempted-recon*.

On profiling the network traffic based on each individual protocol, following results were conclusive:

Traffic Profile by Protocol

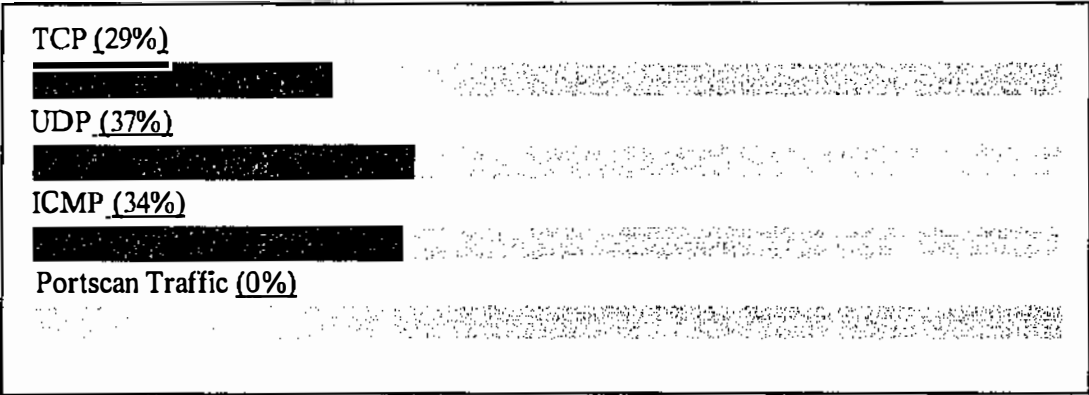


Figure 9.2: Traffic Profile by Protocol using ACID

As illustrated in the figure 9.1 that majority of the network traffic was targeted to UDP (37%) and then ICMP (34%) and TCP (29%). There was also large number of port scans reported on the network. These port numbers can be further detailed on the basis of the occurrences of total alerts and unique alerts on them

Port Type	Occurrences	Unique Alerts
80 /tcp	7777	217
161 /tcp	1375	4
3128 /tcp	1062	2
8080 /tcp	731	2
1080 /tcp	463	1
162 /tcp	407	3
705 /tcp	169	1
0 /udp	139	7
137 /udp	81	77
177 /udp	18	1
10080 /udp	17	1
10081 /udp	16	1
7001 /udp	16	1
22 /tcp	14	8
69 /udp	13	2

Table 9.3: List of popular destination port numbers with occurrence of alerts

From the above table 9.3 it appears that there were 3 ports which reported the maximum number of unique alerts. Those were:

- **80 /tcp:** It is an http port which reported the maximum number of alerts (7777) as well 217 unique alerts. These unique alerts were mainly attack attempt of exploiting WEB-MISC, WEB-CGI and WEB-IIS. These alerts occurred on 64 different hosts connected on the network.
- **137 /udp:** This port number is associated with NetBIOS Name Service. There were 77 unique alerts reported on this port which were mainly different port scans. These attack attempts were targeted on 58 different hosts on the network from 2 different attacking or source hosts. NetBIOS is mainly associated with input and output services, therefore it appears that the attackers tried to exploit such services using NetBIOS based vulnerabilities.
- **22 /tcp:** This is an SSH port used for secure and encrypted transmission of data. This port reported the second most unique alerts i.e. 10. Those alerts were mainly port scans and Nmap scans. These attack attempts were targeted on 191 hosts on the network running port 22. It appears that the attacker tried to exploit this port for gathering any information travelling between the hosts.

9.3 Findings from log files

From the study of log file, it was identified that most of the attackers used Nessus (<http://www.nessus.org>) as the network scanning tool to identify the vulnerabilities in the network. On the findings of the vulnerabilities most of them tried to conduct brute force attacks over the network. There were various probes for exploiting the vulnerabilities in web based services and ports. There were evidence in the log files which shows that the attackers tried to connect few machines to gain access to C:\> drive:

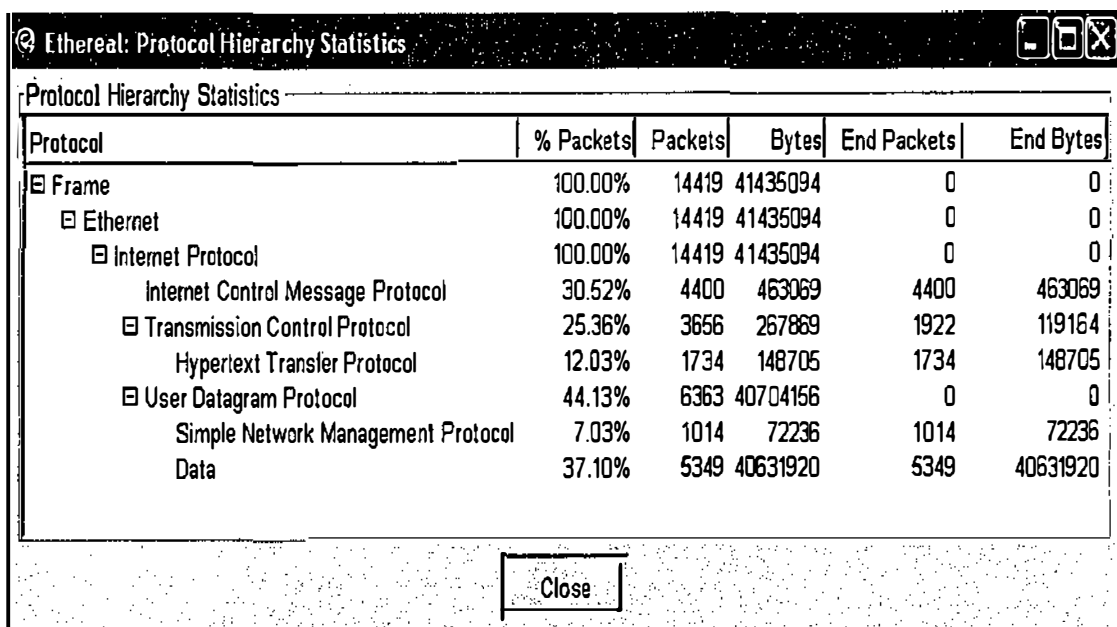
```
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c: HTTP/1.0
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:+/s HTTP/1.0
:
:
:
http://192.168.1.28/scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c
:+/s
http://192.168.0.1/scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:
+/s
http://192.168.1.28/scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c
:+/s
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:+/s HTTP/1.0
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:wwwroot+/s
HTTP/1.0
GET /scripts/..%255c../winnt/system32/cmd.exe?/c+dir+c:wwwroot+/s
HTTP/1.0
```

The above log file entry indicates that the attackers tried to access the file system of IIS server by executing scripts through console based commands. This attack attempt, if successful, could have resulted in providing full access to the web-server to the attackers. This may have resulted in complete compromise of the web-server as the attackers could halt or disable all the web-based services which may have had a big impact on the complete network.

9.4 Ethereal Analysis

There were two tcpdump log files generated during the second testing phase (*tcpdump.log.1053944062* and *tcpdump.log.1054023963*), which was analysed using

Ethereal Packet sniffer. Below is the Protocol Hierarchy statistic generated by the Ethereal:



Protocol	% Packets	Packets	Bytes	End Packets	End Bytes
Frame	100.00%	14419	41435094	0	0
Ethernet	100.00%	14419	41435094	0	0
Internet Protocol	100.00%	14419	41435094	0	0
Internet Control Message Protocol	30.52%	4400	463069	4400	463069
Transmission Control Protocol	25.36%	3656	267869	1922	119164
Hypertext Transfer Protocol	12.03%	1734	148705	1734	148705
User Datagram Protocol	44.13%	6363	40704156	0	0
Simple Network Management Protocol	7.03%	1014	72236	1014	72236
Data	37.10%	5349	40631920	5349	40631920

Figure 9.3: Protocol Hierarchy Statistics of tcpdump.log.1053944062

As shown in the figure 9.2, there were 14419 packets reported in the tcpdump log file *tcpdump.log.1053944062* out of which nearly 30.52% were ICMP packets and 25.36 % TCP packets. Majority of the TCP packets were http packets i.e. 12.03% of the total TCP packets. About 44.13% of total packets consist of UDP packets from which 37.10% were Data packets. This suggests that there was huge amount of data flooded over the network in terms of UDP packets. On further analysing the log file, it was found that there were some frequent occurrences of packets over some of the ports. Those results are listed below:

- A very large amount of UDP Data packets were transmitted on host 192.168.1.28 from source port number 3074 towards port 80 (http port).

0020	01 1c 0c 02 00 50 1e 24 94 e6 2a 2a 2a 2a 2a 20P.\$..,*****
0030	55 44 50 20 46 6c 6f 6f 64 2e 20 53 65 72 76 65	UDP Flood. Serve
0040	72 20 73 74 72 65 73 73 20 74 65 73 74 20 2a 2a	r stress test **
0050	2a 2a 2a 2a 2a 2a 2a 2a 20 55 44 50 20 46 6c 6f	***** UDP Flo
0060	6f 64 2e 20 53 65 72 76 65 72 20 73 74 72 65 73	od. Serv er stres
0070	73 20 74 65 73 74 20 2a 2a 2a 2a 2a 2a 2a 2a	s test * *****
0080	2a 20 55 44 50 20 46 6c 6f 6f 64 2e 20 53 65 72	* UDP Flood. Ser
0090	76 65 72 20 73 74 72 65 73 73 20 74 65 73 74 20	ver stress test

Figure 9.4: UDP Packet

This resulted in the complete delay of the network traffic flow on this particular host which as a whole degraded the performance of network traffic flow on the entire network.

- As mentioned in earlier section 9.3, Ethereal captured the following packet which shows that the attacker ran a script command to retrieve the directory level information:

0000	00 90 2f 1b 94 3a 00 10 d/ 09 e3 a5 08 00 45 00	..'.... ..E.
0010	00 74 2c 8e 40 00 80 06 5f 12 ac 10 01 0f c0 a8	.t,.@... ..
0020	01 1c 0c ac 00 50 80 a0 87 74 ad 62 1a 88 50 18P.. .t.b..P.
0030	40 e8 7b 1e 00 00 47 45 54 20 2f 73 63 72 69 70	@.{...GE T /scrip
0040	74 73 2f 2e 2e 25 32 35 35 63 2e 2e 2f 77 69 6e	ts/..%25 5c../win
0050	6e 74 2f 73 79 73 74 65 6d 33 32 2f 63 6d 64 2e	nt/syste m32/cmd.
0060	65 78 65 3f 2f 63 2b 64 69 72 2b 63 3a 5c 77 77	exe?/c+d ir+c:\ww
0070	77 72 6f 6f 74 2b 2f 73 20 48 54 54 50 2f 31 2e	wroot+/s HTTP/1.
0080	30 0a	0.

Figure 9.5: Packet Captured showing script command used to retrieve information

Ethereal: Protocol Hierarchy Statistics					
Protocol Hierarchy Statistics					
Protocol	% Packets	Packets	Bytes	End Packets	End Bytes
[-] Frame	100.00%	4082	379823	0	0
[-] Ethernet	100.00%	4082	379823	0	0
[-] Internet Protocol	100.00%	4082	379823	0	0
Internet Control Message Protocol	49.73%	2030	185200	2030	185200
[-] Transmission Control Protocol	35.96%	1468	145423	660	47880
Hypertext Transfer Protocol	19.79%	808	97543	808	97543
[-] User Datagram Protocol	14.31%	584	49200	0	0
Simple Network Management Protocol	12.47%	509	42627	509	42627
Data	1.03%	42	4212	42	4212
Internet Security Association and Key Management Protocol	0.07%	3	537	3	537
X Display Manager Control Protocol	0.44%	18	1080	18	1080
Trivial File Transfer Protocol	0.29%	12	744	12	744

Figure 9.6: Protocol Hierarchy Statistics of tcpdump.log. 1054023963

As shown in the figure 9.5, there were 4082 packets reported in the tcpdump log file *tcpdump.log.1054023963* out of which nearly 49.73% were ICMP packets and 35.96% TCP packets. Majority of the TCP packets were *http* packets i.e. 19.79% of the total TCP packets. About 14.31% of total packets consist of UDP packets out of which 12.47% packets were of Simple Network Management Protocol (SNMP).

The above collected data confirms the findings of ACID as Ethereal was able to capture the packets which clearly show the use of UDP Flooder to attempt a Denial of Service (DoS) type of attack. It also captured the packets which illustrate the use of command based script using GET method for retrieving directory level information on the hosts. There were also a reduced number of any other kinds of probes or network scans as in compared to the previous two testings conducted by the participants. There was also no evidence of attempting any remote login access to the network as compared to the previous testing.

9.5 Analyst Notebook 6 graphs

The log files generated by honeyd were imported into the Analyst Notebook 6 to generate the graph of relationship between various hosts on the network. The graph displays various source hosts connecting to destination hosts with linking them according to their occurrences on the network. Since the generated graphs were large in size therefore, they are included on the provided CD along with the chart viewer program. See appendix E for chart file details.

9.6 Implications

After the improvisation of the network, it appears that the level of deception was improved greatly as the level of attack attempts were reduced, especially the TCP network traffic reduced to a considerable extent as compared to the first test results. But one of the prominent attack attempt noticed during the final testing was the attempt of flooding a host with UDP packets to cause DoS based attacks. Other than UDP attack, web-based and ssh based attacks have been consistent on the network. Therefore, it can be assume that the web-based attack attempts are more in use by attackers to exploit vulnerabilities. This may be due to the reason that most of the organisations these days are using web-based services for their day to day work within the organisation.

The current network architecture was greatly improved since the previous 2 network configurations. The only issue which needed attention for improvement was how to tackle UDP based attacks which may be done by dropping the UDP packet requests on the network. There was evidence in the results that one of the attackers tried to carry out a Denial of Service attack by flooding the host with large amount of UDP packets. Otherwise the network traffic behaviour was normal and no more major improvements were required.

On the basis of the results obtained in the final testing, the research believes that the level of deception has improved to the furthest extent as compared to the initial network configuration. There was very little evidence found during the testing which would have created any suspicion within the attackers while performing the penetration exercise.

10. Discussion and Conclusions

The purpose of this research was to enhance the level of deception presented to attackers in honeynet network architecture. Hardening of deceptive honeypots was required to test its effectiveness in gathering attack intelligence using empirical learning approach. The hacking exercise conducted in this research was carried out by 6 out of 10 selected students of School of Computer and Information Science, Edith Cowan University, Western Australia. These students were selected using a Computer Security Literacy Test, a questionnaire with 20 questions related to computer security. The network penetration exercise was conducted at three different occasions. Data collected from each exercise was initially first analysed and the network configuration was improved before the start of next penetration exercise for data collection purpose. After the completion of each exercise, the participants voluntarily provided their feedback report detailing their understanding of the designed network. This process helped in determining the improvement in the level of deception presented to the attackers which obviously was unknown to them.

This research can be divided into 3 different phases: Initial set-up phase, data collection phase and data analysing and summarising phase. There were various challenges and problems encountered during each of these phases.

During the start-up phase of this research, the researcher possessed little knowledge about the honeypots and honeynet. Extensive literature review was conducted to understand the underlying concept of deception and honeypots in today's computer security environment. This research was started just 2 months after the Honeyd was first released in April 2002. There was very little supportive literature available during that period. Therefore, the researcher had to make an attempt to personally communicate with the Honeyd developer, Neil Provos, through emails. Supportive literature, provided by Provos, assisted in understanding the concept of Honeyd and also suggestions made by him helped in installing and maintaining the program which was another tedious task. There were various challenges faced while installing and configuring the honeypot using honeyd in the laboratory network. Various file dependencies were encountered while installing the programs and setting up the

network. Once the installation of honeyd was complete, the implementation of the program using the signature file of various operating systems and network devices (described in detail in Chapters 2 & 5) was required. This NMAP signature file was provided with more than 400 TCP/IP stack fingerprints. There was no valid proof to support that these fingerprints would definitely work with honeyd in emulating the fingerprinted systems. Therefore, Operating system fingerprinting testing was conducted by the researcher by scanning each individual signature, configured in Honeyd, with a network scanning program called Nmap. The results from this testing were that 50% of the total signatures were only valid and successful in emulating the operating systems and network devices. These valid signatures were used as the benchmark for the research. If the researcher had simply assumed that the signatures provided with the honeyd to be 100% valid than there could have been various instances in which the designed honeypot may not have proved to be a success. A lot of time and effort was invested before the actual data collection phase started in establishing and testing the honeypot. Aside from all these obstacles faced during the initial phase of the research, it was a continuous learning experience. By the time data collection stage started, the researcher was well equipped with the greater understanding and knowledge about honeypots and their implementations than had previously been the case.

Data collection was conducted in 3 different stages. Vast amount of data were collected from each stage and was securely stored on a database and also backup stored on a remote machine. The amount of data was enormous therefore it was compressed and stored remotely.

From the data collected from first network penetration exercise it was found that 50% of the network traffic was TCP traffic while UDP and ICMP constituted 18% and 32% respectively. The majority of attacks were targeted at TCP/IP based protocols for example, on http port 80 or 8080 for web based attacks. There were lot of miscellaneous web based attacks on the network out which the most prominent were Proxy scan and WEB-IIS script access. These attacks were among the top 5 alerts identified after the penetration exercise. Out of the 32% network traffic of ICMP, 30% of its traffic was based on ICMP redirect host attack. This was the topmost alert among all the attacks. This type of attack can be very dangerous as it is capable of

bringing down a host machine. Although from the general feedback received from the participants it was suggested that it was a very basic network architecture which lacked the services like FTP or telnet for providing any remote access to the network. According to the participants, remote access for organisational networks is sometime essential as sometime any member of the organisation may not be physically present at organisation but need to access the data remotely. Also according to them the network lacked any server type architecture. Therefore, taking their recommendations into consideration network architecture was improved with the emulated Windows NT server architecture and also by providing remote access to hosts using ftp and telnet scripts to emulate the respective services. These scripts were provided with the honeyd.

Once the network was improved after amending the configuration scripts and upgrading the honeyd with the latest version, the second round of penetration was conducted. It was found that TCP traffic over the network dropped by 5% to 45%. This suggested that the honeypot was more hardened than previous setting and configuration as it allowed less and only legitimate TCP/IP traffic. There was also a minor reduction in UDP traffic too. But the ICMP traffic rose by 4% to 38%. The second network penetration test was more intensive than the previous testing as it provided lot of opportunity for the attackers to explore. The implementation of emulated remote access services like ftp and telnet proved to be successful as there was evidence of attempt of login remotely on to the network. This network penetration testing provided with much more data than before to analyse. There were evidence of use of various tools such as Putty-Release-0.53b (SSH client). Among the top 5 alerts, other than ICMP redirect host and SNMP request udp, there was alert which illustrate that there were attempts of launching web based attacks at the emulated Cisco IOS Router configuration. From the general feedback received from the participants it was believed that other than a few small security risks, the network was quite well designed and properly secured as it did not provide much of opportunity for the attackers to do. According to them, there were only those services running on the network which are common on any network. The file access permissions were properly allocated as the attackers had only read access but did not have any write access to the file systems. The use of a second router provided a better security architecture to the subnet of the network as it allowed only the legitimate

network traffic to the subnet. This router, in addition to its basic functionality of routing the network traffic also did the job of a firewall by blocking malicious traffic. However, from the researcher's point of view it provided large amount of valuable data to study and learn about various types of attacks. From the feedback reports there was evidence that the attackers were confused within the network configuration. As mentioned in one of the report, the attacker got confused with the network entry point believing into that there were 2 hosts sharing the same IP address both located on either side of the network gateway. Therefore, from all these evidences it may be believed that the deceptive honeynet architecture was well appropriate and successful in deceiving the attackers and was also successful in data collection and attack intelligence gathering. Hence, to further verify these results the network configuration was improved and a third and final network penetration exercise was conducted. From the feedback of one of the attacker it was identified that hosts which were operating on windows98 were running IISv5. This is not possible as IISv5 only works on windows 2000, XP or NT platforms. Therefore the script which was emulating the IISv5 was modified to emulate IISv4 on Windows98. This was the only minor change required in the honeypot configuration before the third and final network penetration exercise began.

In the third network penetration test, the amount of TCP traffic was considerably reduced. From the total network traffic, TCP only constituted 29% of the traffic, dropped by 16%, while the UDP traffic increased by 20% to 37%. On analysing these results it was found that this time the attackers tried to launch Denial of Service (DoS) attack by using a Trojan called UDP Flooder to flood the host with large amount of UDP packets. There was not much change in the ICMP traffic. There were also attempts to connect to the emulated Microsoft IIS server using some GET script commands. Such attempts illustrate that the attackers were forced to believe that they were examining a legitimate network with legitimate real services. Therefore, it is reasonable to conclude that the attempt to create network deception using Honeyd was a successful attempt and was able to provide with lot of interesting facts and results which assisted in improving the deception.

After the completion of data collection stage, all the data was securely stored on backup devices like zip disks, cds, and was also stored remotely on a different server.

This data was then analysed using various tools such as ACID, Ethereal and Analyst. There were quite a few problems faced by the researcher during the analysis process of the data. The amount of data collected by honeyd was large. It would not have been possible to analysing that data without the use of software tools such as Analyst. Although Analyst did find it difficult to import that data into its allocated memory in order to visualise the log files. There were several occasions when the machine running Analyst crashed while handling the log files. Therefore, the system memory was upgraded before carrying out any further analysis using Analyst. ACID was a very effective analysing utility. It categorised all the data according to their identity and sources. ACID proved to be a big time saver to researcher while analysing the data.

However, there were few limitations with this honeynet implementation. Due to the restricted nature of this research, it was not possible for the researcher to place this honeynet live onto the internet for the purpose of being getting probed or attacked. This would have provided with more legitimate data about various attacking techniques and tools. This may have also allowed the researcher to explore some new fronts in the honeynet implementation, such as more about data control and data capture, which may be unknown to the researcher. Therefore, this could be one of the tasks which can be carried forward for further research in future.

The configured honeynet was a well implemented network which was able to show various things which a honeypot is capable of doing. With the use of various analysing tools such as ACID, Ethereal and Snort it was possible to visualise this huge amount of data and control the systems. Configuring and implementing a honeynet is a very time consuming and risky task. A small mistake while configuring and implementing a honeynet may cause a serious concern over the network. All the data need to be analysed carefully and proper backup mechanism should be in place. Therefore, a honeynet should be configured with proper guidance and supervision.

However, the honeynet configured for this research can be considered as a successful honeynet as it was possible to successfully deceive the attackers and collect the useful attack intelligence.

11. References

Agassi, J. B. (1992). "Change Research or Action Research: a Promising Methodological Tool, that Combines Applied Sociology with Empirical Research in Organizations." Methodology and Science 25(4): p 196-203.

Anonymous (1996). ISO/OSI Network Model. Available at URL: http://www.uwsg.iu.edu/usail/network/nfs/network_layers.html. Accessed on 11th November 2002.

Anonymous (2000). Honeypot Effectiveness Study, Global Integrity Corporation. Available at URL: www.enetrex.co.kr/support/file%5CHoneypot.pdf. Accessed on 22nd September 2002

Anonymous (2002a). Know Your Enemy, 1st Edition, Published by Addison Wesley.

Anonymous (2002b). Honeynet Project. Available at URL: <http://project.honeynet.org>. Accessed on 15th June 2002.

Anonymous (n.d). ICMP sweep - a whatis definition. Available at URL: http://whatis.techtarget.com/definition/0,,sid9_gci802721,00.html Accessed on 11th February 2003.

Anonymous (n.d(a)). Deception Tool Kit. Available at URL: <http://www.all.net/dtk>. Accessed on 15th June 2002.

Anonymous (n.d(b)). MySQL Reference Manual, MySql AB. Available at URL: http://www.mysql.com/documentation/mysql/bychapter/manual_Introduction.html. Accessed on 10th September 2002.

Anonymous (n.d(c)). Worms, Webopedia. Available at URL: <http://www.webopedia.com/TERM/w/wormn.html>. Accessed on 1st June 2003.

Anonymous (n.d(d)). Advantages and Disadvantages of Experimental Research: Discussion, Writing Center at Colorado State University. Available at URL: <http://writing.colostate.edu/references/research/experiment/pop5c.cfm>. Accessed on 30th June 2003.

Anonymous (n.d(e)). SPECTER Intrusion Detection System. Available at URL: <http://www.specter.com/default50.htm> Accessed on 11th November 2002.

Anonymous (n.d(f)). The Ethereal Network Analyzer. Available at URL: <http://www.ethereal.com> Accessed on 11th December 2002.

Baumann, R. and C. Plattner (2002). Honeypots. School of Computer Science, Diploma Thesis, Swiss Federal Institute of Technology: 143.

Bellovin, S. M. (n.d). Security Problems in the TCP/IP Protocol Suite, AT & T Bell Laboratories. Available at URL: http://www.ja.net/CERT/Bellovin/TCP-IP_Security_Problems.html. Accessed on 11th April 2003.

Bowyer, J. B. (1982). Cheating: deception in war & magin, games & sports, sex & religion, business & con games, politics & espionage, art & science. 1st Edition. New York, St. Martin's Press.

Brenton, C. (n.d). Honeynets, Dartmouth College Institute for Security Technology Studies (ISTS). Available at URL: http://www.ists.dartmouth.edu/IRIA/knowledge_base/honeynets.htm, Accessed on 1st June 2002.

Carthy, R. (1972). Protective coloration and mimicry: nature's camouflage. Westover Publishing Co., New York

Cavaye, A. L. M. (1996). "Case study research: a multi-faceted research approach for IS." Information Systems Journal 6(3): 227-242.

Ceccez-Kecmanovic, D. (2001). Doing Critical IS Research: The Question of Methodology, Idea Group Publishing. Available at URL: www.sistm.unsw.edu.au/people/DUBRAVKA/06dubrav.pdf. Accessed on 30th June 2003.

Cheswick, B. (n.d). An Evening with Berferd, AT & T Bell Laboratories. Available at URL: <http://cne.gmu.edu/modules/acmpkp/security/texts/CRACKER.PDF>, Accessed on 1st June 2002.

Chua, W. F. (1986). "Radical Developments in Accounting Thought." The Accounting Review 61(4): 601-632.

Cohen, F. (1992). Operating System Protection Through Program Evolution Computers and Security. Available at URL: <http://www.all.net/books/IP/evlove.html>. Accessed on 9th March 2003.

Cohen, F. (1996). "Internet Holes - Internet Lightning Rods." Available at URL: <http://www.all.net/journal/netsec/1996-07-2.html>, Accessed on 4th June 2002.

Cohen, F. (1998). "A note on the role of deception in information protection." Computers & Security 17(6): 483.

Cohen, F. (2000a). "A mathematical structure of simple defensive network deceptions." Computers & Security 19(6): 520.

Cohen, F. (2000b). "The Structure of Intrusion and Intrusion Detection." Available at URL: <http://www.all.net/journal/ntb/IDSstructure.html>, Accessed on 29th May 2002.

Cohen, F., D. Lambert, C. Preston, N. Berry, C. Stewart and E. Thomas. (2001). A Framework for Deception. Available at URL: <http://www.all.net/journal/deception/Framework/Framework.html>. Accessed on 11th November 2002.

Comer, D. E. (1995). Internetworking with TCP/IP - Principles, Protocols and Architecture. Volume 1, 3rd Edition. New Jersey, Prentice Hall.

Comer, D. E. and D. L. Stevens (1999). Internetworking with TCP/IP - Design, Implementation and Internals. Volume 2. 3rd Edition. New Jersey, Prentice Hall.

Creswell, J. W. (2003). Research design: qualitative, quantitative and mixed method approaches. 2nd Edition, London, Sage Publications.

Daly, J., J. Miller, A. Brooks, M. Roper and M. Wood (1995). "A Multi-Method Approach to Performing Empirical Research." Systems and Software.

Danyliw, R. (n.d). Analysis Console for Intrusion Databases (ACID). Available at URL: <http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html> Accessed on 18th July 2002.

Doran, R. (2000). Deep sea anglerfish. Available at URL: http://ramseydoran.com/anglerfish/deep_sea.htm. Accessed on 10th April 2003.

Dunnigan, J. and A. A. Nofi (1995). Victory and Deceit - Dirty Tricks at War. New York, Published by William Morrow and Co.

Fairhurst, G. (2001). Address Resolution Protocol (ARP). Available at URL: <http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/arp.html>. Accessed on 19th September 2002.

Farrow, R. (2000). System Fingerprinting with Nmap, Network Magazine. November Edition. Available at URL: <http://www.networkmagazine.com/article/NMG20001102S0005>. Accessed on 2nd December 2002.

Fowler, C. A. and R. F. Nesbit (1995). "Tactical Deception in air-land warfare." Journal of Electronic Defense 18(6): 37-44 & 76-79.

Fujita, F. (1996). The Big Five Taxonomy. Available at URL: <http://www.iusb.edu/~ffujita/Documents/ecology.html>. Accessed on 4th April 2003.

Fyodor (n.d). Nmap -- Free Stealth Port Scanner for Network Exploitation & Security Audits. Available at URL: <http://www.insecure.org/nmap> Accessed on 11th December 2002.

Fyodor (1998). Remote OS detection via TCP/IP Stack Fingerprinting. Available at URL: <http://www.insecure.org/nmap/nmap-fingerprinting-article.txt>. Accessed on 15th March 2002.

Galliers, R. D. and F. E. Land (1987). Choosing an appropriate information systems research methodologies. Communicaitons of the ACM, **30**: 900-902.

Gerwehr, S. (n.d). National Security: Lessons from animal and plant deceptions. Available at URL: http://www.rand.org/natsec_area/products/animal.html. Accessed on 1st September 2002.

Gerwehr, S. and R. H. Anderson (2000). Employing deception in INFOSEC. Available at URL: <http://www.cert.org/research/isw/isw2000/papers/26.pdf>. Accessed on 11th April 2003.

Gerwehr, S. and R. W. Glenn (2000). The Art of Darkness: Deception and Urban Operations, RAND Publication.

Glaser, T. (2000). TCP/IP Stack Fingerprinting Principles, SANS Institute resources. Available at URL: http://www.sans.org/newlook/resources/IDFAQ/TCP_fingerprinting.htm. Accessed on 22nd August 2002.

Godson, R. and J. J. Wirtz (2002). Strategic Denial And Deception. New Jersey, Transaction Publishers.

Greenhalgh, T. and R. Taylor (n.d). How to read a qualitative research paper. Available at URL: http://dse212.port5.com/qualitative_methods.htm. Accessed on 3rd April 2003.

Gupta, A. (2003). Knowledge share in high technology sectors with an aim to assess the impact of decision making on the outcomes of New Product Development, Unpublished doctoral thesis, University of Nottingham, UK.

Haveeru, D. (2003). The hunt for red octopus, Haveeru daily. Available at URL: <http://www.haveeru.com.mv/english/features/octopus.html>. Accessed on 14th March 2003.

Hirani, S., M. Ali, B. Duenas, Y. Stryker and G. Chim. (n.d). Denial of Service, Royal Holloway University of London: 25. Available at URL: <http://www.isg.rhul.ac.uk/msc/teaching/ic4/2002/groups/Group09.doc>. Accessed in 2003

Holcroft, S. (2002). Design Of a Default RedHat Server 6.2 Honeypot. Available at URL: <http://www.lucidic.net/whitepapers/sholcroft-4-2002.html>. Accessed on 30th May 2002.

Hutchinson, B. and M. Warren (2002). Deception in Cyberspace. Available at URL: http://www.dlux.org.au/dataterra/deception_in_cyberspace.html. Accessed on 16th January 2003.

Judd, C. M., E. R. Smith and L. H. Kidder (1991). Research Methods in Social Relations. Florida, Published by Harcourt Brace Jovanovich, Inc., USA.

Klug, D. (2000). Honey Pots and Intrusion Detection. Available at URL: <http://rr.sans.org/intrusion/honeypots.php>. Accessed on 30th May 2002.

Kruglanski, A. W. (1975). The human subject in the psychology experiment: Fact and Artifact. In L. Berkowitz (Ed.) Advances in experimental social psychology. Vol. 8, New York, Academic Press.

Lewis, M. and C. Saami (1993). Lying and Deception in Everyday Life. New York, The Guilford Press.

Low, C. (2001). ICMP Attacks Illustrated. Available at URL: http://www.sans.org/rr/threats/ICMP_attacks.php. Accessed on 23rd April 2003.

Monterey Bay Aquarium (2003). Anglerfish: Melanocetus johnsoni, Available at URL: http://www.mbayaq.org/efc/living_species/default.asp?hOri=1&inhab=176. Accessed on 10th April 2003.

Myers, M. D. (1997). Qualitative Research in Information Systems. MIS Quarterly. 21: pp 241-242.

NORMAN (2003). Attacks against weaknesses in the TCP/IP protocol, Norman. Available at URL: http://www.norman.com/documents/wp_smurf.shtml. Accessed on 11th April 2003.

Orlikowski, W. J. and J. J. Baroudi (1991). "Studying Information Technology in Organizations: Research Approaches and Assumptions." Information Systems Research 2(1): 1-28.

Parker, T. and M. Sportack (2000). TCP/IP Unleashed, Sams Publishing.

Provos, N. (2002). Honeyd - Network Rhapsody For You. Available at URL: <http://www.citi.umich.edu/u/provos/honeyd/>. Accessed on 12th June 2002.

Provos, N. (2002 (a)). libevent - an event notification library. Available at URL: <http://www.monkey.org/~provos/libevent/> Accessed on 9th December 2002.

Provos, N. (2003). Honeyd: A Virtual Honeypot Daemon. Available at URL: <http://www.citi.umich.edu/u/provos/honeyd/>. Accessed on 31st March 2003.

Polar Bears International (2002). Polar bear fur. Available on <http://www.polarbearsalive.org/facts3.htm#768453>, Accessed on 12th March 2003

Rodriguez, A., J. Gatrell, J. Karas and R. Peshcke (2001). TCP/IP Tutorial and Technical Overview, 7th Edition, IBM Corporation.

Roesch, M. (n.d). Snort - Lightweight Intrusion Detection for Networks. Available at URL: <http://www.snort.org/docs/lisapaper.txt>. Accessed on 28th November 2002.

Rose, G. (1982). Deciphering Sociological Research. 1st Edition, London, Macmillan.

Rowe, N. C. and H. Rothstein (2003). Deception for Defense of Information System: Analogies from Conventional Warfare. Available at URL: <http://www.cs.nps.navy.mil/people/faculty/rowe/mildec.htm>. Accessed on 1st May 2003.

Sarantakos, S. (1993). Social Research. 1st Editon. Melbourne, Macmillan Education Australia Pty Ltd.

Scharge, M. (1999). The tangled Web of e-deception. Fortune. 140:6 (296).

Scheidler, B. (1999). Syslog-ng reference manual. Available at URL: <http://www.balabit.hu/static/syslog-ng/reference/book1.html>. Accessed on 1st October 2002.

Scottberg, B., W. Yurcik, and D. Doss. (2002). Internet Honeypots: Protection or Entrapment. IEEE International Symposium on Technology and Society (ISTAS), Raleigh NC, USA, IEEE Service Center.

Sink, M. (2001). The use of Honeypots and Packet Sniffers for Intrusion Detection. Available at URL: http://rr.sans.org/intrusion/honey_pack.php. Accessed on 29th May 2002.

Snog, D. (n.d). libdnet. Available at URL: <http://libdnet.sourceforge.net/> Accessed on 9th December 2002.

Spitzner, L. (2002). Honeypots. Available at URL: <http://www.enteract.com/~lspitz/honeypot.html>. Accessed on 5th June 2002.

Spitzner, L. (2003). Honeypots: Tracking Hackers. Boston, Addison Wesley.

Steven, W. R. (1994). TCP/IP Illustrated, Vol. 1. Addison Wesley Publishing Company.

Sundaram, A. (2001). "An Introduction to Intrusion Detection." ACM Crossroads. Available at URL: www.acm.org/crossroads/xrds2-4/intrus.html, Accessed on 15th May 2002.

Webster's Revised Unabridged Dictionary. Dictionary.com/deception. Available at URL: <http://www.dictionary.com/search?q=deception>. Accessed on 23rd August 2002.

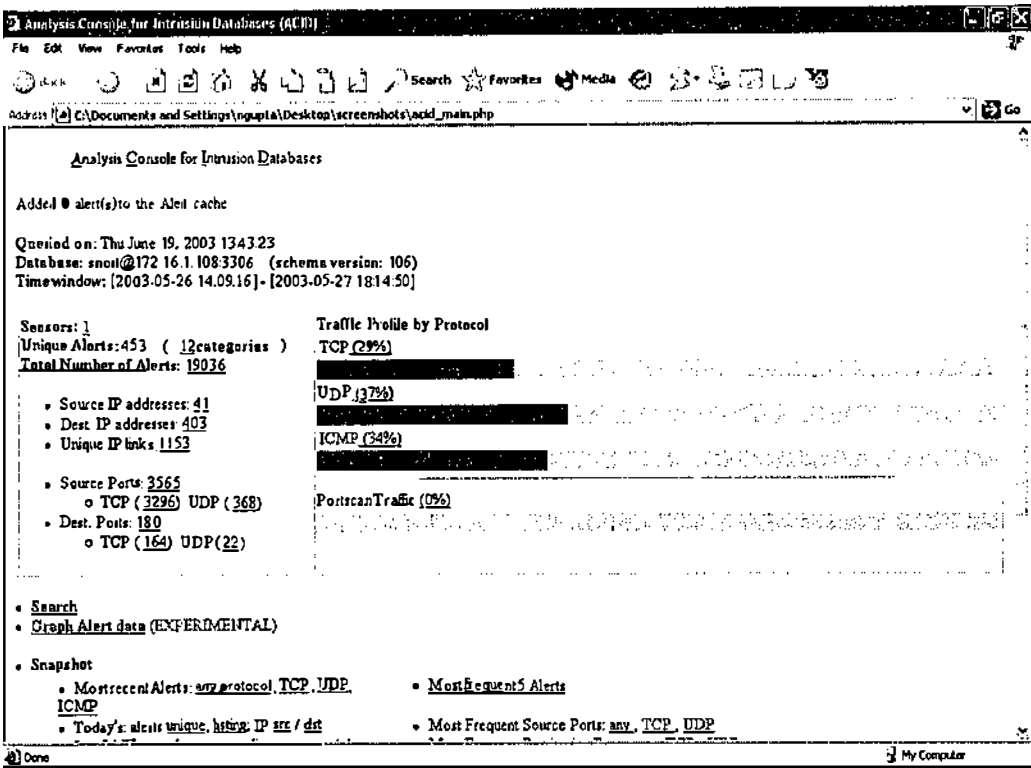
Whaley, B. (1969). Stratagem: Deception and Surprise in War, Cambridge: MIT Center for International Studies.

Wood, M., J. Daly and M. Roper (1998). "Multi-Method Research: An Empirical Investigation of Object-Oriented Technology." Systems and Software. Available at URL: <http://www.cis.strath.ac.uk/research/papers/EFoCS-12-95.Z>, Accessed on 8th January 2003

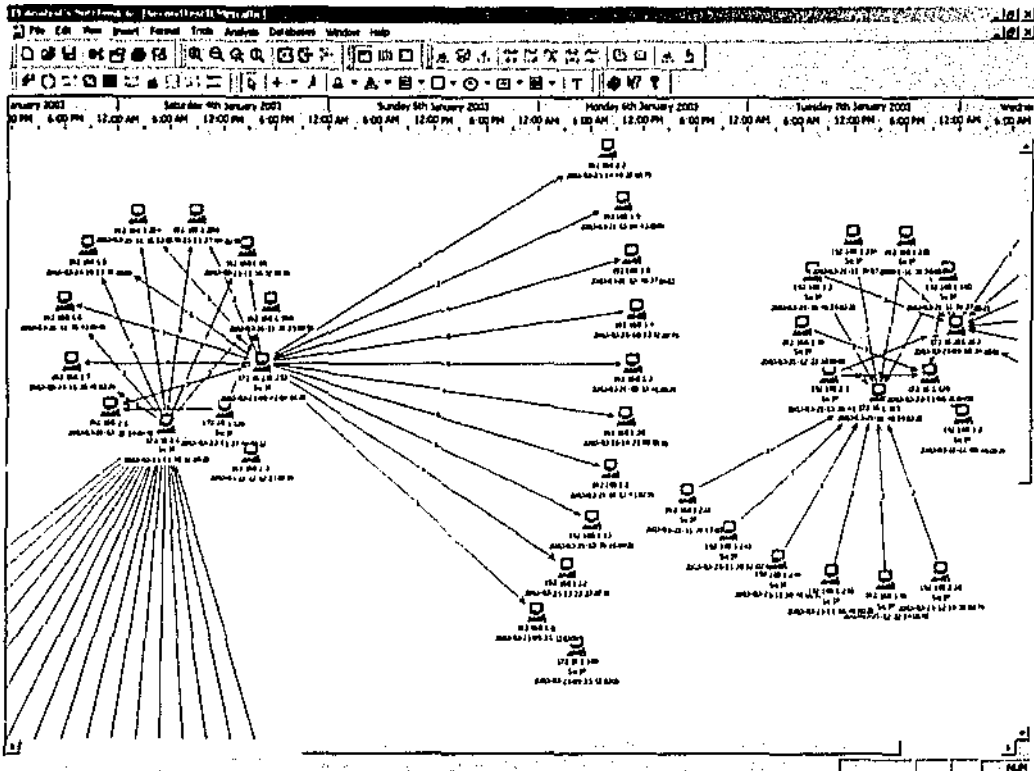
Appendix A

Screenshots

A.1 ACID



A.2 Analyst Notebook 6



A.3 GFI LANguard

The screenshot shows the GFI LANguard Network Security Scanner interface. The left pane displays a tree view of the scan results for the target IP 10.82.24.107. The right pane shows the detailed scan results for this IP.

Left Pane (Tree View):

- Network Security Scanner (GFI)
 - NEIGHBOR (1)
 - Username: NGUPTA
 - MAC: 00:02:B3:08:3C:4D (Intel Corporation)
 - Time to live (TTL): 128 (128) - Same network segment
 - LAN Manager, Windows 2000 LAN Manager
 - Domain: ADS
 - Computer usage: Workstation
 - Trusted domains (1)
 - Shares (3)
 - Groups (10)
 - Users (7)
 - Services (47)
 - Sessions (1)
 - Network devices (2)
 - Local drives (4)
 - Remote TOD (Time of day)
 - Password policy
 - Registry
 - Installed patches (11)
 - Security audit policy
 - TCP Ports (7)
 - UDP Ports (6)
 - Alerts (47)
 - Missing Security Patches/Services (27)
 - Windows XP Professional Gold
 - Internet Explorer 5 Gold
 - Office XP Service Pack 1
 - Internet Information Services 5.1 Gold
 - Patches which cannot be detected (10)
 - Service Alerts (4)
 - Administrative account events
 - User KUSR_CS000283085060 (Internet Guest Account) never logged on
 - User SUPPORT_1_00694540 (CN=Microsoft Corporation, Redmond WA) never logged on
 - User YUSR_CS000283085060 (VSA Server Account) never logged on
 - Registry Alerts (5)
 - AutoShareServer (1)
 - AutoShareWKS (2)

Right Pane (Scan Results):

```

HITRIS discovery ...
Done sending, waiting for responses ...
Reply from 10.82.24.107 (HITRIS)
SNMP discovery ...
Community string : public
Done sending, waiting for responses ...
ICMP sweep ... (PING)
Done sending, waiting for responses ...
PONG from 10.82.24.107
Time to live (TTL) : 128 (128)
Same network segment
ICMP code in response = 0 -> Windows box
Timestamp Reply (10.82.24.107)
ICMP unreachable for closed port (10.82.24.107)
Ready
1 Computer(s) found.
[10.82.24.107]
SNMP probing ...
Connecting ... (1/6)
Name "HITRIS" encoded as
"INEJEEDEHFFAFEEBCACACACACACAC"
Session established.(2/6)
Security mode : user
Protocol negotiated.(3/6)
Operating system : Windows XP
Domain : ADS
LAN manager : Windows 2000 LAN Manager
HITL session established.(4/6)
Connected to IPCS.(5/6)
No share list.

Read server info ...
List trusted domains ...
List shares ...
List groups ...
List users ...
List services ...
List sessions ...
List network transports ...
List drives ...
Read remote time of day ...
  
```

A.4 Ethereal

The screenshot shows the Ethereal (Wireshark) interface with a packet capture of ICMP Redirect messages. The top pane displays a list of 16 packets, all of which are ICMP Redirects from 172.16.1.1 to 172.16.1.13. The middle pane shows the details of the selected packet (No. 1), including the Ethernet II header, Internet Protocol header, and Internet Control Message Protocol (ICMP) details. The bottom pane shows the raw packet data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Info
2	1.003217	172.16.1.1	172.16.1.13	ICMP	Redirect
3	1.003046	172.16.1.1	172.16.1.13	ICMP	Redirect
4	615.973990	172.16.1.105	172.16.1.1	TCP	1251 > 3128 [SYN] Seq=4208704236 Ack=0 Win=16384 Len=0
5	941.228343	172.16.1.1	172.16.1.13	ICMP	Redirect
6	942.286105	172.16.1.1	172.16.1.13	ICMP	Redirect
7	942.733183	172.16.1.1	172.16.1.13	ICMP	Redirect
8	943.285450	172.16.1.1	172.16.1.13	ICMP	Redirect
9	943.301805	172.16.1.1	192.168.1.1	ICMP	Redirect
10	943.371780	172.16.1.1	172.16.1.13	ICMP	Redirect
11	944.234475	172.16.1.1	172.16.1.13	ICMP	Redirect
12	946.048149	172.16.1.1	172.16.1.13	ICMP	Redirect
13	946.061798	172.16.1.1	192.168.1.1	ICMP	Redirect
14	946.549834	172.16.1.1	172.16.1.13	ICMP	Redirect
15	947.091783	172.16.1.1	192.168.1.1	ICMP	Redirect
16	948.067809	172.16.1.1	172.16.1.13	ICMP	Redirect

Frame 1 (126 bytes on wire (126 bytes captured))
 Ethernet II, Src: 08:00:27:1b:94:7a, Dst: 08:10:d7:09:e3:a5
 Internet Protocol, Src Addr: 172.16.1.1 (172.16.1.1), Dst Addr: 172.16.1.13 (172.16.1.13)
 Internet Control Message Protocol
 Type: 3 (Redirect)
 Code: 1 (Redirect for host)
 Checksum: 03355 (correct)
 Gateway Address: 192.168.0.1
 Internet Protocol, Src Addr: 172.16.1.11 (172.16.1.11), Dst Addr: 192.168.0.1 (192.168.0.1)
 Internet Control Message Protocol

A.5 Webmin

The screenshot shows the Webmin web interface, which is a web-based system configuration tool. The interface includes a navigation bar with links to various system management tasks, a search bar, and a footer with copyright information.

Webmin - Microsoft Internet Explorer

Address: http://www.webmin.com/

Navigation links: Home, Search, About, Contact, Feedback

Webmin - System Configuration

Released: 11 May 2003
 Download: tar.gz | RPM

Webmin

Introduction to Webmin
 Updates to Webmin
 Change Log
 Development Versions of Webmin

Supported Operating Systems
 Standard Modules
 Bug Tracking System
 Wishlist for new Projects

Downloading and Installing
 Supported Languages
 Author's Wishlist
 Webmin and Usermin Graphics

Screenshots
 Third-Party Modules
 Custom Webmin Development

A.6 Nmap

File		Output		Help	
Host(s): xanadu vectra playground				Scan	Exit
Scan Options:		General Options:			
<input checked="" type="checkbox"/> connect()	<input type="checkbox"/> Don't Resolve	<input checked="" type="checkbox"/> TCP Ping	<input type="checkbox"/> Fragmentation		
<input checked="" type="checkbox"/> SYN Stealth	<input type="checkbox"/> Fast Scan	<input checked="" type="checkbox"/> TCP&ICMP	<input type="checkbox"/> Get Identd Info		
<input checked="" type="checkbox"/> Ping Sweep	<input type="checkbox"/> Range of Ports:	<input checked="" type="checkbox"/> ICMP Ping	<input type="checkbox"/> Resolve All		
<input checked="" type="checkbox"/> UDP Port Scan		<input checked="" type="checkbox"/> Don't Ping	<input checked="" type="checkbox"/> OS Detection		
<input checked="" type="checkbox"/> FIN Stealth	<input type="checkbox"/> Use Decoy(s):	<input type="checkbox"/> Input File:	<input type="checkbox"/> Send on Device:		
<input checked="" type="checkbox"/> Bounce Scan:	antionline.com				
Output from: nmap -sS -O -Dantionline.com xanadu vectra playground					
Interesting ports on vectra.yuma.net (192.168.0.5):					
Port	State	Protocol	Service		
13	open	tcp	daytime		
21	open	tcp	ftp		
22	open	tcp	ssh		
23	open	tcp	telnet		
37	open	tcp	time		
79	open	tcp	finger		
111	open	tcp	sunrpc		
113	open	tcp	auth		
513	open	tcp	login		
514	open	tcp	shell		
TCP Sequence Prediction: Class=random positive increments Difficulty=14943 (Worthy challenge)					
Remote operating system guess: OpenBSD 2.2 - 2.3					
Interesting ports on playground.yuma.net (192.168.0.1):					
Port	State	Protocol	Service		

Appendix B

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

I. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section I above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program,

the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include

anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any

other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not pennit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is pennitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have

the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY

OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.> Copyright (C)
19yy <name of author>
```

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>,

1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to pennit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix C

Scripts

C.1 honeyd.conf(version Honeyd0.4a, used for 1st Test)

```
route entry 10.11.68.18
route 10.11.68.18 link 10.11.68.0/24
route 10.11.68.18 add net 10.11.69.0/24 10.11.68.1 latency 55ms loss
0.1
route 10.11.68.1 link 10.11.69.0/24

create mars
set mars personality "Windows 2000 Professional, Build 2128"
add mars tcp port 80 "sh scripts/web.sh"
set mars default tcp action reset
bind 10.11.69.1 mars

create mercury
set mercury personality "AIX 3.2"
add mercury tcp port 80 "sh scripts/web.sh"
set mercury default tcp action reset
bind 10.11.69.2 mercury

create earth
set earth personality "Solaris 2.3 - 2.4"
add earth tcp port 80 "sh scripts/web.sh"
set earth default tcp action reset
bind 10.11.69.3 earth

create venus
set venus personality "FreeBSD 3.2-4.0"
add venus tcp port 80 "sh scripts/web.sh"
set venus default tcp action reset
bind 10.11.69.4 venus

create jupiter
set jupiter personality "Cisco Router/Switch with IOS 11.2"
add jupiter tcp port 80 "sh scripts/web.sh"
set jupiter default tcp action reset
bind 10.11.68.18 jupiter

create sun
set sun personality "Cisco 760 Series (non IOS) or IBM Stackable Hub"
add sun tcp port 80 "sh scripts/web.sh"
set sun default tcp action reset
bind 10.11.68.19 sun

create pluto
set pluto personality "Novell NetWare 3.12 or 386 TCP/IP"
add pluto tcp port 80 "sh scripts/web.sh"
set pluto default tcp action reset
bind 10.11.68.10 pluto
```



```

create moon
set moon personality "Windows 98"
add moon tcp port 80 "sh scripts/web.sh"
set moon default tcp action reset
bind 10.11.68.11 moon

create default
set default personality "Windows 98"
set default default tcp action reset
add default tcp port 80 "sh scripts/web.sh"
add default tcp port 22 "sh scripts/test.sh"
add default tcp port 139 open

```

C.2 honeyd.conf (version honeyd05a, used for 2nd and 3rd Tests)

```

## Honeyd configuration file ##

#####
#
##### ROUTING CONFIGURATION
#####
#####

route entry 192.168.1.100
route 192.168.1.100 link 192.168.1.0/24
route 192.168.1.100 add net 192.168.2.0/24 192.168.1.1 latency 55ms
loss 0.1
route 192.168.1.1 link 192.168.2.0/24

#####
#
##### SYSTEM CONFIGURATION
#####
#####

### Windows computers
create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
set windows default tcp action reset
set windows default udp action reset
add windows tcp port 80 "perl scripts/iis-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows uptime 3284460
bind 192.168.2.101 windows

### AIX computer
create aix
set aix personality "AIX 3.2"
set aix default tcp action reset
add aix tcp port 25 block

```

```
add aix tcp port 80 "sh scripts/web.sh"
add aix tcp port 21 "sh scripts/ftp.sh"
set aix uptime 3284460
bind 192.168.2.102 aix
```

```
### Solaris computers
create solaris
set solaris personality "Solaris 2.3 - 2.4"
add solaris tcp port 80 "sh scripts/web.sh"
set solaris default tcp action reset
bind 192.168.2.103 solaris
```

```
### FreeBSD computers
create bsd
set bsd personality "FreeBSD 3.2-4.0"
add bsd tcp port 80 "sh scripts/web.sh"
set bsd default tcp action reset
bind 192.168.2.104 bsd
```

```
### Cisco router
create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
set router default tcp action reset
set router default udp action reset
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 1327650
bind 192.168.1.100 router
```

```
### Cisco hub
create hub
set hub personality "Cisco Router/Switch with IOS 11.2"
set hub default tcp action reset
set hub default udp action reset
add hub tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set hub uptime 1327650
bind 192.168.1.101 hub
```

```
### Novell Computers
create novell
set novell personality "Novell Netware 5.0 SP5"
add novell tcp port 80 "sh scripts/web.sh"
set novell default tcp action reset
bind 192.168.1.115 novell
```

```
### Windows 98 computers
create win98
set win98 personality "Windows 98"
add win98 tcp port 80 "sh scripts/web98.sh"
set win98 default tcp action reset
bind 192.168.1.116 win98
```

```
### Default computers
create default
set default personality "Windows 98"
set default default tcp action reset
add default tcp port 80 "sh scripts/web98.sh"
add default tcp port 22 "sh scripts/test.sh"
add default tcp port 139 open
```

C.3 web.sh

```
#!/bin/sh
REQUEST=""
while read name
do
    LINE=`echo "$name" | egrep -i "[a-z:]"`
    if [ -z "$LINE" ]
    then
        break
    fi
    echo "$name" >> /tmp/log
    NEWREQUEST=`echo "$name" | grep "GET .scripts.*cmd.exe.*dir.*
HTTP/1.0"`
    if [ ! -z "$NEWREQUEST" ] ; then
        REQUEST=$NEWREQUEST
    fi
done

if [ -z "$REQUEST" ] ; then
    cat << _eof_
HTTP/1.1 404 NOT FOUND
Server: Microsoft-IIS/5.0
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo
OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI'
Content-Location: http://cpmsftwbw27/default.htm
Date: Thu, 04 Apr 2002 06:42:18 GMT
Content-Type: text/html
Accept-Ranges: bytes

<html><title>You are in Error</title>
<body>
<h1>You are in Error</h1>
O strange and inconceivable thing! We did not really die, we were not
really buried, we were not really crucified and raised again, but our
imitation was but a figure, while our salvation is in reality. Christ
was actually crucified, and actually buried, and truly rose again;
and all these things have been vouchsafed to us, that we, by
imitation communicating in His sufferings, might gain salvation in
reality. O surpassing loving-kindness! Christ received the nails in
His undefiled hands and feet, and endured anguish; while to me
without suffering or toil, by the fellowship of His pain He
vouchsafed salvation.
<p>
St. Cyril of Jerusalem, On the Christian Sacraments.
</body>
</html>
_eof_
    exit 0
fi

DATE=`date`
cat << _eof_
HTTP/1.0 200 OK
Date: $DATE
Server: Microsoft-IIS/5.0
Connection: close
```

Content-Type: text/plain

Volume in drive C is Webserver
Volume Serial Number is 3421-07F5
Directory of C:\inetpub

```
01-20-02    3:58a        <DIR>
08-21-01    9:12a        <DIR>          ..
08-21-01   11:28a        <DIR>          AdminScripts
08-21-01    6:43p        <DIR>          ftproot
07-09-00   12:04a        <DIR>          iissamples
07-03-00    2:09a        <DIR>          mailroot
07-16-00    3:49p        <DIR>          Scripts
07-09-00    3:10p        <DIR>          webpub
07-16-00    4:43p        <DIR>          wwwroot
           0 file(s)                0 bytes
           20 dir(s)            290,897,920 bytes free
_eof_
```

C.4 web98.sh

```
#!/bin/sh
REQUEST=""
while read name
do
    LINE=`echo "$name" | egrep -i "[a-z:]"`
    if [ -z "$LINE" ]
    then
        break
    fi
    echo "$name" >> /tmp/log
    NEWREQUEST=`echo "$name" | grep "GET .scripts.*cmd.exe.*dir.*
HTTP/1.0"`
    if [ ! -z "$NEWREQUEST" ] ; then
        REQUEST=$NEWREQUEST
    fi
done

if [ -z "$REQUEST" ] ; then
    cat << _eof_
HTTP/1.1 404 NOT FOUND
Server: Microsoft-IIS/4.0
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo
OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI'
Content-Location: http://cpmsftwbw27/default.htm
Date: Thu, 04 Apr 2002 06:42:18 GMT
Content-Type: text/html
Accept-Ranges: bytes

<html><title> Error</title>
<body>

The parameter is incorrect.
</body>
</html>
_eof_
    exit 0
```

fi

```
DATE=`date`
cat << _eof_
HTTP/1.0 200 OK
Date: $DATE
Server: Microsoft-IIS/4.0
Connection: close
Content-Type: text/plain
```

```
Volume in drive C is Webserver
Volume Serial Number is 2A24-150C
Directory of C:\inetpub
```

```
01-20-02    1:00a        <DIR>          .
08-21-01   11:15a        <DIR>
08-21-01   11:55a        <DIR>          wwwroot
08-21-01   11:56a        <DIR>          iissamples
08-21-01   11:56a        <DIR>          scripts
08-21-01   11:56a        <DIR>          webpub
                0 file(s)                0 bytes
                6 dir(s)          311,671,296 bytes free
_eof_
```

C.5 snort.conf

```
#-----
#   http://www.snort.org      Snort 1.9.0 Ruleset
#   Contact: snort-sigs@lists.sourceforge.net
#-----
# NOTE: This ruleset only works for 1.9.0 and later
#-----
# $Id: snort.conf,v 1.110 2002/08/14 03:17:58 chrisgreen Exp $
#
#####
# This file contains a sample snort configuration.
# You can take the following steps to create your
# own custom configuration:
#
#   1) Set the network variables for your network
#   2) Configure preprocessors
#   3) Configure output plugins
#   4) Customize your rule set
#
#####
# Step #1: Set the network variables:
#
# You must change the following variables to reflect
# your local network. The variable is currently
# setup for an RFC 1918 address space.
#
# You can specify it explicitly as:
#
# var HOME_NET 10.1.1.0/24
#
# or use global variable $<interfacename>_ADDRESS
# which will be always initialized to IP address and
```

```

# netmask of the network interface which you run
# snort at.
#
# var HOME_NET $eth0_ADDRESS
#
# You can specify lists of IP addresses for HOME_NET
# by separating the IPs with commas like this:
#
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
#
# or you can specify the variable to be any IP address
# like this:

var HOME_NET any

# Set up the external network addresses as well.
# A good start may be "any"

var EXTERNAL_NET any

# Configure your server lists. This allows snort to only look for
attacks
# to systems that have a service up. Why look for HTTP attacks if
you are
# not running a web server? This allows quick filtering based on IP
addresses
# These configurations MUST follow the same configuration scheme as
defined
# above for $HOME_NET.

# List of DNS servers on your network
var DNS_SERVERS $HOME_NET

# List of SMTP servers on your network
var SMTP_SERVERS $HOME_NET

# List of web servers on your network
var HTTP_SERVERS $HOME_NET

# List of sql servers on your network
var SQL_SERVERS $HOME_NET

# List of telnet servers on your network
var TELNET_SERVERS $HOME_NET

# Configure your service ports. This allows snort to look for
attacks
# destined to a specific application only on the ports that
application
# runs on. For example, if you run a web server on port 8081, set
your
# HTTP_PORTS variable like this:
#
# var HTTP_PORTS 8081
#
# Port lists must either be continuous [eg 80:8080], or a single port
[eg 80].
# We will adding support for a real list of ports in the future.

```

```

# Ports you run web servers on
var HTTP_PORTS 80

# Ports you want to look for SHELLCODE on.
var SHELLCODE_PORTS !80

# Ports you do oracle attacks on
var ORACLE_PORTS 1521

# other variables
#
# AIM servers. AOL has a habit of adding new AIM servers, so instead
of
# modifying the signatures when they do, we add them to this list of
# servers.
var AIM_SERVERS
[64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.0/
24,64.12.161.0/24,64.12.163.0/24,205.188.5.0/24,205.188.9.0/24]

# Path to your rules files (this can be a relative path)
var RULE_PATH /etc/snort

#####:#####
# Step #2: Configure preprocessors
#
# General configuration for preprocessors is of
# the form
# preprocessor <name_of_processor>: <configuration_options>

# frag2: IP defragmentation support
# -----
# This preprocessor performs IP defragmentation. This plugin will
also detect
# people launching fragmentation attacks (usually DoS) against hosts.
No
# arguments loads the default configuration of the preprocessor,
which is a
# 60 second timeout and a 4MB fragment buffer.

# The following (comma delimited) options are available for frag2
#   timeout [seconds] - sets the number of [seconds] than an
unfinished
#                               fragment will be kept around waiting for
completion,
#                               if this time expires the fragment will be
flushed
#   memcap [bytes] - limit frag2 memory usage to [number] bytes
#                               (default: 4194304)
#
#   min_ttl [number] - minimum ttl to accept
#
#   ttl_limit [number] - difference of ttl to accept without
alerting
#                               will cause false positives with router flap
#
# Frag2 uses Generator ID 113 and uses the following SIDS
# for that GID:
#   SID      Event description
#   -----
#   1        Oversized fragment (reassembled frag > 64k bytes)
#   2        Teardrop-type attack

```

preprocessor frag2

```
# stream4: stateful inspection/stream reassembly for Snort
#-----
--
# Use in concert with the -z [all|est] command line switch to defeat
# stick/snot against TCP rules. Also performs full TCP stream
# reassembly, stateful inspection of TCP streams, etc. Can
statefully
# detect various portscan types, fingerprinting, ECN, etc.

# stateful inspection directive
# no arguments loads the defaults (timeout 30, memcap 8388608)
# options (options are comma delimited):
#   detect_scans - stream4 will detect stealth portscans and generate
alerts
#                   when it sees them when this option is set
#   detect_state_problems - detect TCP state problems, this tends to
be very
#                   noisy because there are a lot of crappy
ip stack
#                   implementations out there
#
#   disable_evasion_alerts - turn off the possibly noisy mitigation
of
#                   overlapping sequences.
#
#
#   min_ttl [number] - set a minium ttl that snort will accept
to
#                   stream reassembly
#
#   ttl_limit [number] - differential of the initial ttl on a
session versus
#                   the normal that someone may be playing
games.
#                   Routing flap may cause lots of false
positives.
#
#   keepstats [machine|binary] - keep session statistics, add
"machine" to
#                   get them in a flat format for machine
reading, add
#                   "binary" to get them in a unified binary
output
#                   format
#   noinspect - turn off stateful inspection only
#   timeout [number] - set the session timeout counter to [number]
seconds,
#                   default is 30 seconds
#   memcap [number] - limit stream4 memory usage to [number] bytes
#   log_flushed_streams - if an event is detected on a stream this
option will
#                   cause all packets that are stored in the
stream4
#                   packet buffers to be flushed to disk. This
only
#                   works when logging in pcap mode!
#
# Stream4 uses Generator ID 111 and uses the following SIDS
```



```

# for that GID:
#  SID      Event description
#  -----  -
#  1        Stealth activity
#  2        Evasive RST packet
#  3        Evasive TCP packet retransmission
#  4        TCP Window violation
#  5        Data on SYN packet
#  6        Stealth scan: full XMAS
#  7        Stealth scan: SYN-ACK-PSH-URG
#  8        Stealth scan: FIN scan
#  9        Stealth scan: NULL scan
#  10       Stealth scan: NMAP XMAS scan
#  11       Stealth scan: Vecna scan
#  12       Stealth scan: NMAP fingerprint scan stateful detect
#  13       Stealth scan: SYN-FIN scan
#  14       TCP forward overlap

preprocessor stream4: detect_scans, disable_evasion_alerts

# tcp stream reassembly directive
# no arguments loads the default configuration
#   Only reassemble the client,
#   Only reassemble the default list of ports (See below),
#   Give alerts for "bad" streams
#
# Available options (comma delimited):
#   clientonly - reassemble traffic for the client side of a
# connection only
#   serveronly - reassemble traffic for the server side of a
# connection only
#   both - reassemble both sides of a session
#   noalerts - turn off alerts from the stream reassembly stage of
# stream4
#   ports [list] - use the space separated list of ports in [list],
# "all"
#
#                       will turn on reassembly for all ports, "default"
will turn
#                       on reassembly for ports 21, 23, 25, 53, 80, 143,
110, 111
#                       and 513

preprocessor stream4_reassemble

# http_decode: normalize HTTP requests
# -----
# http_decode normalizes HTTP requests from remote
# machines by converting any %XX character
# substitutions to their ASCII equivalent. This is
# very useful for doing things like defeating hostile
# attackers trying to stealth themselves from IDSs by
# mixing these substitutions in with the request.
# Specify the port numbers you want it to analyze as arguments.
#
# Major code cleanups thanks to rfp
#
# unicode           - normalize unicode
# iis_alt_unicode   - %u encoding from iis
# double_encode     - alert on possible double encodings
# iis_flip_slash    - normalize \ as /
# full_whitespace   - treat \t as whitespace ( for apache )

```

```

#
# for that GID:
#   SID      Event description
#   -----  ~~~~~
#   1        UNICODE attack
#   2        NULL byte attack

preprocessor http_decode: 80 unicode iis_alt_unicode double_encode
iis_flip_slash full_whitespace

# rpc_decode: normalize RPC traffic
# -----
# RPC may be sent in alternate encodings besides the usual
# 4-byte encoding that is used by default.  This preprocessor
# normalized RPC traffic in much the same way as the http_decode
# preprocessor.  This plugin takes the ports numbers that RPC
# services are running on as arguments.
# The RPC decode preprocessor uses generator ID 106 and does not
# generate any SIDs at this time.

preprocessor rpc_decode: 111 32771

# bo: Back Orifice detector
# -----
# Detects Back Orifice traffic on the network.  This preprocessor
# uses the Back Orifice "encryption" algorithm to search for
# traffic conforming to the Back Orifice protocol (not B02K).
# This preprocessor can take two arguments.  The first is "-nobrute"
# which turns off the plugin's brute forcing routine (brute forces
# the key space of the protocol to find B0 traffic).  The second
# argument that can be passed to the routine is a number to use
# as the default key when trying to decrypt the traffic.  The
# default value is 31337 (just like B0).  Be aware that turning on
# the brute forcing option runs the risk of impacting the overall
# performance of Snort, you've been warned...
#
# The Back Orifice detector uses Generator ID 105 and uses the
# following SIDS for that GID:
#   SID      Event description
#   -----  ~~~~~
#   1        Back Orifice traffic detected

preprocessor bo: -nobrute

# telnet_decode: Telnet negotiation string normalizer
# -----
# This preprocessor "normalizes" telnet negotiation strings from
# telnet and ftp traffic.  It works in much the same way as the
# http_decode preprocessor, searching for traffic that breaks up
# the normal data stream of a protocol and replacing it with
# a normalized representation of that traffic so that the "content"
# pattern matching keyword can work without requiring modifications.
# This preprocessor requires no arguments.
# Portscan uses Generator ID 109 and does not generate any SID
# currently.

preprocessor telnet_decode

# Portscan: detect a variety of portscans
# -----
# portscan preprocessor by Patrick Mullen <p_mullen@linuxrc.net>

```

```

# This preprocessor detects UDP packets or TCP SYN packets going to
# four different ports in less than three seconds. "Stealth" TCP
# packets are always detected, regardless of these settings.
# Portscan uses Generator ID 100 and uses the following SIDS for that
# GID:
#   SID      Event description
#   -----
#   1        Portscan detect
#   2        Inter-scan info
#   3        Portscan End

# preprocessor portscan: $HOME_NET 4 3 portscan.log

# Use portscan-ignorehosts to ignore TCP SYN and UDP "scans" from
# specific networks or hosts to reduce false alerts. It is typical
# to see many false alerts from DNS servers so you may want to
# add your DNS servers here. You can all multiple hosts/networks
# in a whitespace-delimited list.
#
#preprocessor portscan-ignorehosts: 0.0.0.0

# arpspoof
#-----
# Experimental ARP detection code from Jeff Nathan, detects ARP
# attacks,
# unicast ARP requests, and specific ARP mapping monitoring. To make
# use
# of this preprocessor you must specify the IP and hardware address
# of hosts on # the same layer 2 segment as you. Specify one host IP
# MAC combo per line.
# Also takes a "-unicast" option to turn on unicast ARP request
# detection.
# Arpspoof uses Generator ID 112 and uses the following SIDS for that
# GID:
#   SID      Event description
#   -----
#   1        Unicast ARP request
#   2        Etherframe ARP mismatch (src)
#   3        Etherframe ARP mismatch (dst)
#   4        ARP cache overwrite attack

#preprocessor arpspoof
#preprocessor arpspoof_detect_host: 192.168.40.1 f0:0f:00:f0:0f:00

# ASN1 Decode
#-----
# This is an experimental preprocessor. ASN.1 decoder and analysis
# plugin
# from Andrew R. Baker. This preprocessor will detect abuses of the
# ASN.1
# protocol that higher level protocols (like SSL, SNMP, x.509, etc)
# rely on.
# The ASN.1 decoder uses Generator ID 115 and uses the following SIDS
# for
# that GID:
#   SID      Event description
#   -----
#   1        Indefinite length
#   2        Invalid length
#   3        Oversized item
#   4        ASN.1 specification violation

```

```

# 5          Dataum bad length

preprocessor asnl_decode

# Fnord
#-----
# This is an experimental preprocessor. Polymorphic shellcode
# analyzer and
# detector by Dragos Ruiu. This preprocessor will watch traffic for
# polymorphic NOP-type sleds to defeat tools like ADMutate. The
# Fnord detector
# uses Generator ID 114 and the following SIDs:
#   SID      Event description
#   ----      -
#   1         NOP-sled detected

# preprocessor fnord

# Conversation
#-----
# This preprocessor tracks conversations for tcp, udp and icmp
# traffic. It
# is a prerequisite for running portscan2.
#
# allowed_ip_protocols 1 6 17
#       list of allowed ip protocols ( defaults to any )
#
# timeout {num}
#       conversation timeout ( defaults to 60 )
#
#
# max_conversations {num}
#       number of conversations to support at once (defaults to 65335)
#
#
# alert_odd_protocols
#       alert on protocols not listed in allowed_ip_protocols

preprocessor conversation: allowed_ip_protocols all, timeout 60,
max_conversations 32000

# Portscan2
#-----
# Portscan 2, detect portscans in a new and exciting way.
#
# Available options:
#       scanners_max {num}
#       targets_max {num}
#       target_limit {num}
#       port_limit {num}
#       timeout {num}
#       log [logdir]

preprocessor portscan2: scanners_max 3200, targets_max 5000,
target_limit 5, port_limit 20, timeout 60

# Experimental Perf stats
# -----
# No docs. Highly subject to change.
#
# preprocessor perfmonitor: console flow events time 10

```

```

#####
# Step #3: Configure output plugins
#
# Uncomment and configure the output plugins you decide to use.
# General configuration for output plugins is of the form:
#
# output <name_of_plugin>: <configuration_options>
#
#alert_syslog: log alerts to syslog
# -----
# Use one or more syslog facilities as arguments
#
# output alert_syslog: LOG_AUTH LOG_ALERT

# log_tcpdump: log packets in binary tcpdump format
# -----
# The only argument is the output file name.
#
# output log_tcpdump: tcpdump.log

# database: log to a variety of databases
# -----
# See the README.database file for more information about configuring
# and using this plugin.
#
# output database: alert, mysql, user=snort password=03piggy03
# dbname=snort host=localhost detail=full
# output database: alert, postgresql, user=snort dbname=snort
# output database: log, unixodbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test

# xml: xml logging
# -----
# See the README.xml file for more information about configuring
# and using this plugin.
#
# output xml: log, file=/var/log/snortxml

# unified: Snort unified binary format alerting and logging
# -----
# The unified output plugin provides two new formats for logging
# and generating alerts from Snort, the "unified" format. The
# unified format is a straight binary format for logging data
# out of Snort that is designed to be fast and efficient. Used
# with barnyard (the new alert/log processor), most of the overhead
# for logging and alerting to various slow storage mechanisms
# such as databases or the network can now be avoided.
#
# Check out the spo_unified.h file for the data formats.
#
# Two arguments are supported.
#   filename - base filename to write to (current time_t is
# appended)
#   limit    - maximum size of spool file in MB (default: 128)
#
# output alert_unified: filename snort.alert, limit 128
# output log_unified: filename snort.log, limit 128

# trap_snmp: SNMP alerting for Snort
# -----

```

```

# Read the README.SNMP file for more information on enabling and
using this
# plug-in.
#
#

#The trap_snmp plugin accepts the following notification options
# [c], [p[m|s]]
# where,
#     c : Generate compact notifications. (Saves on bandwidth by
#         not reporting MOs for which values are unknown, not
#         available or, not applicable). By default this option is
reset.
#     p : Generate a print of the invariant part of the offending
packet.
#         This can be used to track the packet across the Internet.
#         By default this option is reset.
#     m : Use the MD5 algorithm to generate the packet print.
#         By default this algorithm is used.
#     s : Use the SHA1 algorithm to generate the packet print.
#
# The trap_snmp plugin requires several parameters
# The parameters depend on the Snmpversion that is used (specified)
# For the SNMPv2c case the parameters will be as follows
# alert, <sensorID>, [NotificationOptions] ,
# {trap|inform} -v <SnmpVersion> -p <portNumber> <hostName>
<community>
#
# For SNMPv2c traps with MD5 digest based packetPrint generation
#
# output trap_snmp: alert, 7, cpm, trap -v 2c myTrapListener
myCommunity
#
# For SNMPv2c informs with the 'compact' notification option
#
# output trap_snmp: alert, 7, c, inform -v 2c myTrapListener
myCommunity
#
#
# For SNMPv3 traps with
# security name = snortUser
# security level = authentication and privacy
# authentication parameters :
#     authentication protocol = SHA ,
#     authentication pass phrase = SnortAuthPassword
# privacy (encryption) parameters
#     privacy protocol = DES,
#     privacy pass phrase = SnortPrivPassword
#
# output trap_snmp: alert, 7, trap -v 3 -u snortUser -l authPriv -a
SHA -A SnortAuthPassword -x DES -X SnortPrivPassword myTrapListener
# For SNMPv3 informs with authentication and encryption
# output trap_snmp: alert, 7, inform -v 3 -u snortUser -l authPriv -a
SHA -A SnortAuthPassword -x DES -X SnortPrivPassword myTrapListener

# You can optionally define new rule types and associate one or
# more output plugins specifically to that type.
#
# This example will create a type that will log to just tcpdump.
# ruletype suspicious
# {

```

```

#   type log
#   output log_tcpdump: suspicious.log
# }
#
# EXAMPLE RULE FOR SUSPICIOUS RULETYPE:
# suspicious $HOME_NET any -> $HOME_NET 6667 (msg:"Internal IRC
Server";)
#
# This example will create a rule type that will log to syslog
# and a mysql database.
# ruletype redalert
# {
#   type alert
#   output alert_syslog: LOG_AUTH LOG_ALERT
#   output database: log, mysql, user=snort dbname=snort
host=localhost
# }
#
# EXAMPLE RULE FOR REDALERT RULETYPE
# redalert $HOME_NET any -> $EXTERNAL_NET 31337 (msg:"Someone is
being LEET"; \
#   flags:A+;)

#
# Include classification & priority settings
#

include classification.config

#
# Include reference systems
#

include reference.config

#####
# Step #4: Customize your rule set
#
# Up to date snort rules are available at http://www.snort.org
#
# The snort web site has documentation about how to write your own
# custom snort rules.
#
# The rules included with this distribution generate alerts based on
# on suspicious activity. Depending on your network environment, your
# security policies, and what you consider to be suspicious, some of
# these rules may either generate false positives ore may be
detecting
# activity you consider to be acceptable; therefore, you are
# encouraged to comment out rules that are not applicable in your
# environment.
#
# Note that using all of the rules at the same time may lead to
# serious packet loss on slower machines. YMMV, use with caution,
# standard disclaimers apply. :)
#
# The following individuals contributed many of rules in this
# distribution.
#
# Credits:
#   Ron Gula <rgula@securitywizards.com> of Network Security Wizards

```

```

# Max Vision <vision@whitehats.com>
# Martin Markgraf <martin@mail.du.gtn.com>
# Fyodor Yarochkin <fygrave@tigerteam.net>
# Nick Rogness <nick@rapidnet.com>
# Jim Forster <jforster@rapidnet.com>
# Scott McIntyre <scott@whoi.edu>
# Tom Vandepoel <Tom.Vandepoel@ubizen.com>
# Brian Caswell <bmc@snort.org>
# Zeno <admin@cgisecurity.com>
# Ryan Russell <ryan@securityfocus.com>
#
#=====
# Include all relevant rulesets here
#
# shellcode, policy, info, backdoor, and virus rulesets are
# disabled by default. These require tuning and maintance.
# Please read the included specific file for more information.
#=====

include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules

include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules

include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/snmp.rules

include $RULE_PATH/smtp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/pop3.rules

include $RULE_PATH/nnntp.rules
include $RULE_PATH/other-ids.rules
# include $RULE_PATH/web-attacks.rules
# include $RULE_PATH/backdoor.rules
# include $RULE_PATH/shellcode.rules
# include $RULE_PATH/policy.rules
# include $RULE_PATH/porn.rules

```



```
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
# include $RULE_PATH/virus.rules
# include $RULE_PATH/chat.rules
# include $RULE_PATH/multimedia.rules
# include $RULE_PATH/p2p.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/local.rules
```

C.6 ftp.sh

```
#!/bin/sh
#
# FTP (WU-FTPD) Honeypot-Script intended for use with
# Honeyd from Niels Provos
# -> http://www.citi.umich.edu/u/provos/honeyd/
#
# Author: Maik Ellinger
# Last modified: 13/06/2002
# Version: 0.0.8
#
# Changelog:
# 0.0.6; some ftp comamnds implemented (MKD)
#
# 0.0.4; some ftp comamnds implemented (CWD)
#
# 0.0.3: some bugfixes/new commands implemented
#
# 0.0.1: initial release
#

#set -x -v
DATE=`date`
host=`hostname`
domain=`dnsdomainname`
log=/tmp/honeyd/ftp-$1.log
AUTH="no"
PASS="no"
echo "$DATE: FTP started from $1 Port $2" >> $log
echo -e "220 $host.$domain FTP server (Version wu-2.6.0(5) $DATE)
ready.\r"
while read incmd parm1 parm2 parm3 parm4 parm5
do
    # remove control-characters
    incmd=`echo $incmd | sed s/[[:cntrl:]]//g`
    parm1=`echo $parm1 | sed s/[[:cntrl:]]//g`
    parm2=`echo $parm2 | sed s/[[:cntrl:]]//g`
    parm3=`echo $parm3 | sed s/[[:cntrl:]]//g`
    parm4=`echo $parm4 | sed s/[[:cntrl:]]//g`
    parm5=`echo $parm5 | sed s/[[:cntrl:]]//g`

    # convert to upper-case
    incmd_nocase=`echo $incmd | gawk '{print toupper($0);}'`
    #echo $incmd_nocase

    if [ "$AUTH" == "no" ]
    then
        if [ "$incmd_nocase" != "USER" ]
```

```

        then
        if [ "$incmd_nocase" != "QUIT" ]
        then
            echo -e "530 Please login with USER and PASS.\r"
            continue
        fi
    fi
fi

case $incmd_nocase in

    QUIT* )
        echo -e "221 Goodbye.\r"
        exit 0;;
    SYST* )
        echo -e "215 UNIX Type: L8\r"
        ;;
    HELP* )
        echo -e "214-The following commands are recognized (*
=>'s unimplemented).\r"
        echo -e "      USER      PORT      STOR      MSAM*      RNT0      NLST
MKD      CDUP\r"
        echo -e "      PASS      PASV      APPE      MRSQ*      ABOR      SITE
XMKD      XCUP\r"
        echo -e "      ACCT*     TYPE      MLFL*     MRCP*     DELE      SYST
RMD      STOU\r"
        echo -e "      SMNT*     STRU      MAIL*     ALLO      CWD       STAT
XRMD      SIZE\r"
        echo -e "      REIN*     MODE      MSND*     REST      XCWD      HELP
PWD      MDTM\r"
        echo -e "      QUIT      RETR      MSOM*     RNFR      LIST      NOOP
XPWD\r"
        echo -e "214 Direct comments to ftp@$domain.\r"
        ;;
    USER* )
        parml_nocase=`echo $parml | gawk '{print toupper($0);}'`
        if [ "$parml_nocase" == "ANONYMOUS" ]
        then
            echo -e "331 Guest login ok, send your complete e-mail
address as a password.\r"
            AUTH="ANONYMOUS"
        else
            echo -e "331 Password required for $parml\r"
            AUTH=$parml
        fi
        ;;
    PASS* )
        PASS=$parml
        if [ "$AUTH" == "ANONYMOUS" ]
        then
            echo -e "230-Hello User at $1,\r"
            echo -e "230-we have 911 users (max 1800) logged in
in your class at the moment.\r"
            echo -e "230-Local time is: $DATE\r"
            echo -e "230-All transfers are logged. If you don't
like this, disconnect now.\r"
            echo -e "230-\r"
            echo -e "230-tar-on-the-fly and gzip-on-the-fly are
implemented; to get a whole\r"
            echo -e "230-directory \"foo\", \"get foo.tar\" or
\"get foo.tar.gz\" may be used.\r"

```

```

        echo -e "230-Please use gzip-on-the-fly only if you
need it; most files already\r"
        echo -e "230-are compressed, and I will kill your
processes if you waste my\r"
        echo -e "230-ressources.\r"
        echo -e "230-\r"
        echo -e "230-The command \"site exec locate pattern\"
will create a list of all\r"
        echo -e "230-path names containing \"pattern\".\r"
        echo -e "230-\r"
        echo -e "230 Guest login ok, access restrictions
apply.\r"
    else
        echo -e "530 Login incorrect.\r"
    fi
    ;;
MKD* )
    # choose :
    echo -e "257 \"$parm1\" new directory created.\r"
    #echo -e "550 $parm1: Permission denied.\r"
    ;;
CWD* )
    # choose :
    echo -e "250 CWD command successful.\r"
    # echo -e "550 $parm1: No such file or directory.\r"
    ;;
NOOP* )
    echo -e "200 NOOP command successful.\r"
    ;;
PASV* )
    echo -e "227 Entering Passive Mode
(134,76,11,100,165,53)\r"
    ;;
ACCT* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
SMNT* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
REIN* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MLFL* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MAIL* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MSND* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MSON* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MSAM* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MRSQ* )
    echo -e "502 $incmd command not implemented.\r"
    ;;
MRCP* )

```

```

        echo -e "502 $incmd command not implemented.\r"
        ;;
    MLFL* )
        echo -e "502 $incmd command not implemented.\r"
        ;;
    * )
        echo -e "500 '$incmd': command not understood.\r"
        ;;
esac
echo -e "$incmd $parm1 $parm2 $parm3 $parm4 $parm5" >> $log
done

```

C.7 create_db.sql

```

-- Copyright (C) 2000-2002 Carnegie Mellon University
--
-- Maintainer: Roman Danyliw <rdd@cert.org>, <roman@danyliw.com>
--
-- Original Author(s): Jed Pickel <jed@pickel.net>      (2000-2001)
--                   Roman Danyliw <rdd@cert.org>
--                   Todd Schrubbs <tls@cert.org>
--
-- This program is free software; you can redistribute it and/or
-- modify
-- it under the terms of the GNU General Public License as published
-- by
-- the Free Software Foundation; either version 2 of the License, or
-- (at your option) any later version.
--
-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
-- GNU General Public License for more details.
--
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-
-- 1307, USA.

CREATE TABLE schema ( vseq          INT4          NOT NULL,
                       ctime         DATETIME NOT NULL,
                       PRIMARY KEY (vseq));
INSERT INTO schema (vseq, ctime) VALUES ('106', now());

CREATE TABLE signature ( sig_id      SERIAL NOT NULL,
                          sig_name    TEXT NOT NULL,
                          sig_class_id INT8,
                          sig_priority INT8,
                          sig_rev     INT8,
                          sig_sid     INT8,
                          PRIMARY KEY (sig_id));
CREATE INDEX sig_name_idx ON signature (sig_name);
CREATE INDEX sig_class_id ON signature (sig_class_id);

CREATE TABLE sig_reference (sig_id INT4 NOT NULL,
                             ref_seq INT4 NOT NULL,
                             ref_id  INT4 NOT NULL,

```

```

PRIMARY KEY(sig_id, ref_seq));

CREATE TABLE reference ( ref_id          SERIAL,
                          ref_system_id INT4 NOT NULL,
                          ref_tag         TEXT NOT NULL,
                          PRIMARY KEY (ref_id));

CREATE TABLE reference_system ( ref_system_id SERIAL,
                                ref_system_name TEXT,
                                PRIMARY KEY (ref_system_id));

CREATE TABLE sig_class ( sig_class_id     SERIAL,
                           sig_class_name   TEXT NOT NULL,
                           PRIMARY KEY (sig_class_id) );
CREATE INDEX sig_class_name_idx ON sig_class (sig_class_name);

CREATE TABLE event ( sid          INT4 NOT NULL,
                      cid          INT8 NOT NULL,
                      signature     INT4 NOT NULL,
                      timestamp     DATETIME NOT NULL,
                      PRIMARY KEY (sid,cid));
CREATE INDEX signature_idx ON event (signature);
CREATE INDEX timestamp_idx ON event (timestamp);

-- store info about the sensor supplying data
CREATE TABLE sensor ( sid          SERIAL,
                      hostname      TEXT,
                      interface     TEXT,
                      filter        TEXT,
                      detail        INT2,
                      encoding       INT2,
                      last_cid      INT8 NOT NULL,
                      PRIMARY KEY (sid));

-- All of the fields of an ip header
CREATE TABLE iphdr ( sid          INT4 NOT NULL,
                      cid          INT8 NOT NULL,
                      ip_src       INT8 NOT NULL,
                      ip_dst       INT8 NOT NULL,
                      ip_ver       INT2,
                      ip_hlen      INT2,
                      ip_tos       INT2,
                      ip_len       INT4,
                      ip_id        INT4,
                      ip_flags     INT2,
                      ip_off       INT4,
                      ip_ttl       INT2,
                      ip_proto     INT2 NOT NULL,
                      ip_csum      INT4,
                      PRIMARY KEY (sid,cid));
CREATE INDEX ip_src_idx ON iphdr (ip_src);
CREATE INDEX ip_dst_idx ON iphdr (ip_dst);

-- All of the fields of a tcp header
CREATE TABLE tcphdr( sid          INT4 NOT NULL,
                      cid          INT8 NOT NULL,
                      tcp_sport    INT4 NOT NULL,
                      tcp_dport    INT4 NOT NULL,
                      tcp_seq      INT8,
                      tcp_ack      INT8,
                      tcp_off      INT2,

```

```

        tcp_res      INT2,
        tcp_flags    INT2 NOT NULL,
        tcp_win      INT4,
        tcp_csum      INT4,
        tcp_urp      INT4,
        PRIMARY KEY (sid,cid));
CREATE INDEX tcp_sport_idx ON tcphdr (tcp_sport);
CREATE INDEX tcp_dport_idx ON tcphdr (tcp_dport);
CREATE INDEX tcp_flags_idx ON tcphdr (tcp_flags);

-- All of the fields of a udp header
CREATE TABLE udphdr(  sid      INT4 NOT NULL,
                      cid      INT8 NOT NULL,
                      udp_sport INT4 NOT NULL,
                      udp_dport INT4 NOT NULL,
                      udp_len   INT4,
                      udp_csum   INT4,
                      PRIMARY KEY (sid,cid));
CREATE INDEX udp_sport_idx ON udphdr (udp_sport);
CREATE INDEX udp_dport_idx ON udphdr (udp_dport);

-- All of the fields of an icmp header
CREATE TABLE icmphdr( sid      INT4 NOT NULL,
                      cid      INT8 NOT NULL,
                      icmp_type INT2 NOT NULL,
                      icmp_code INT2 NOT NULL,
                      icmp_csum  INT4,
                      icmp_id    INT4,
                      icmp_seq   INT4,
                      PRIMARY KEY (sid,cid));
CREATE INDEX icmp_type_idx ON icmphdr (icmp_type);

-- Protocol options
CREATE TABLE opt      ( sid      INT4 NOT NULL,
                      cid      INT8 NOT NULL,
                      optid    INT2 NOT NULL,
                      opt_proto INT2 NOT NULL,
                      opt_code  INT2 NOT NULL,
                      opt_len   INT4,
                      opt_data  TEXT,
                      PRIMARY KEY (sid,cid,optid));

-- Packet payload
CREATE TABLE data      ( sid      INT4 NOT NULL,
                      cid      INT8 NOT NULL,
                      data_payload TEXT,
                      PRIMARY KEY (sid,cid));

-- encoding is a lookup table for storing encoding types
CREATE TABLE encoding(encoding_type INT2 NOT NULL,
                      encoding_text TEXT NOT NULL,
                      PRIMARY KEY (encoding_type));
INSERT INTO encoding (encoding_type, encoding_text) VALUES (0,
'hex');
INSERT INTO encoding (encoding_type, encoding_text) VALUES (1,
'base64');
INSERT INTO encoding (encoding_type, encoding_text) VALUES (2,
'ascii');

-- detail is a lookup table for storing different detail levels
CREATE TABLE detail (detail_type INT2 NOT NULL,

```

```

        detail_text TEXT NOT NULL,
        PRIMARY KEY (detail_type));
INSERT INTO detail (detail_type, detail_text) VALUES (0, 'fast');
INSERT INTO detail (detail_type, detail_text) VALUES (1, 'full');

-- be sure to also use the snortdb-extra tables if you want
-- mappings for tcp flags, protocols, and ports

```

C.8 router-telnet.pl

```

#!/usr/bin/perl
# Copyright 2002 Niels Provos <provos@citi.umich.edu>
# All rights reserved.
#
# For the license refer to the main source code of Honeyd.
#
# Don't echo Will Echo Will Surpress Go Ahead
$return = pack('cccccccc', 255, 254, 1, 255, 251, 1, 255, 251, 3);
syswrite STDOUT, $return, 9;

$string =
"Users (authorized or unauthorized) have no explicit or\r
implicit expectation of privacy. Any or all uses of this\r
system may be intercepted, monitored, recorded, copied,\r
audited, inspected, and disclosed to authorized site,\r
and law enforcement personnel, as well as to authorized\r
officials of other agencies, both domestic and foreign.\r
By using this system, the user consents to such\r
interception, monitoring, recording, copying, auditing,\r
inspection, and disclosure at the discretion of authorized\r
site.\r
\r
Unauthorized or improper use of this system may result in\r
administrative disciplinary action and civil and criminal\r
penalties. By continuing to use this system you indicate\r
your awareness of and consent to these terms and conditions\r
of use. LOG OFF IMMEDIATELY if you do not agree to the\r
conditions stated in this warning.\r
\r
\r
User Access Verification\r
";

syswrite STDOUT, $string;

$count = 0;
while ($count < 3) {
    do {
        $count++;
        syswrite STDOUT, "\r\n";
        $word = read_word("Username: ", 1);
    } while (!$word && $count < 3);
    if ($count >= 3 && !$word) {
        exit;
    }
    $password = read_word("Password: ", 0);
    if (!$password) {

```

```

        syswrite STDOUT, "% Login invalid\r\n";
    } else {
        syswrite STDERR, "Attempted login: $word/$password";
        syswrite STDOUT, "% Access denied\r\n";
    }
}

exit;

sub read_word {
    local $prompt = shift;
    local $echo = shift;
    local $word;

    syswrite STDOUT, "$prompt";

    $word = "";
    $alarmed = 0;
    eval {
        local $SIG{ALRM} = sub { $alarmed = 1; die; };
        alarm 30;
        $finished = 0;
        do {
            $nread = sysread STDIN, $buffer, 1;
            die unless $nread;
            if (ord($buffer) == 0) {
                ; #ignore
            } elsif (ord($buffer) == 255) {
                sysread STDIN, $buffer, 2;
            } elsif (ord($buffer) == 13 || ord($buffer) == 10) {
                syswrite STDOUT, "\r\n" if $echo;
                $finished = 1;
            } else {
                syswrite STDOUT, $buffer, 1 if $echo;
                $word = $word.$buffer;
            }
        } while (!$finished);
        alarm 0;
    };
    syswrite STDOUT, "\r\n" if $alarmed || ! $echo;
    if ($alarmed) {
        syswrite STDOUT, "% $prompt timeout expired!\r\n";
        return (0);
    }
    return ($word);
}

```


Appendix D

Paper Published

Gupta N (2002), *Improving the Effectiveness of Deceptive Honeynets through an Empirical Learning Approach*, Published in Proceedings of 3rd Australian Information Warfare & Security Conference 2002, Perth, Australia

Appendix E

Contents of CD

- Analyst Notebook 6 Chat viewer
- Chart Files:
 - Second Test TCP Traffic
 - Second Test TCP Traffic Cluster
 - Second Test UDP Traffic
 - Second Test UDP Traffic Cluster
 - Second Test ICMP Traffic
 - Second Test ICMP Traffic Cluster

 - Third Test TCP Traffic
 - Third Test TCP Traffic Cluster
 - Third Test UDP Traffic
 - Third Test UDP Traffic Cluster
 - Third Test ICMP Traffic
 - Third Test ICMP Traffic Cluster

How to Install Chart Reader 6

To install *Chart Reader 6*:

1. Close all applications that you may have open.
2. Insert the CD containing *Chart Reader 6* into your computer's CD drive.
3. Browse to the CD, open the Chart Reader folder and run *Setup.exe*.

The Setup.exe is located at:
CD Drive:\Chart Reader
4. Follow the prompts to complete the installation.
5. Run *Chart Reader 6* from your Start menu: **i2 > Chart Reader 6 > Chart Reader 6**